Technische Universität München
Informatik 5 – Lehrstuhl für Wissenschaftliches Rechnen

Partitioned Fluid-Structure Interaction on Massively Parallel Systems

Benjamin Walter Uekermann

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des Akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende(r): Prof. Dr. Dr. Arndt Bode

Prüfer der Dissertation:
1. Prof. Dr. Hans-Joachim Bungartz
2. Prof. Dr. Miriam Mehl, Universität Stuttgart
3. Prof. Dr. Carol Woodward,
   North Carolina State University, USA

Die Dissertation wurde am 30.08.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 12.10.2016 angenommen.

Abstract

Today's multi-physics applications suffer from a lack of either flexibility or scalability. Only reconciling both will allow to translate the higher compute power of the forthcoming exascale era into more complex scenarios with more relevant physical effects covered. Only then, the promised tremendous advances in the most challenging multi-physics applications, such as climate modeling or simulations of the human body, are feasible.

The coupling library preCICE enables the coupling of multiple black-box single-physics solvers at runtime. In my thesis, I introduce parallelization on two levels into preCICE without compromising this flexibility. First, parallelization on an intra-solver level: A new parallel concept renders a central coupling instance unnecessary, but executes all preCICE features on distributed data. Second, parallelization on an inter-solver level: I develop novel parallel quasi-Newton coupling schemes, which show the same convergence speed as state-of-the-art sequential alternatives.

These parallelization concepts allow for a high scalability of complete multi-physics setups. The coupling does no longer degenerate the solvers' scalability significantly. Applications can now be ported from moderately to massively parallel machines. Thus, mesh resolutions can be adjusted for simulations of turbulent fluid-structure interaction, blood-flow with real geometries, or multi-field scenarios such as fluid-structure-acoustics interaction.

Contents

1 Introduction

The forthcoming exa-scale era promises immense computational power. Naturally, there is a strong belief that this comes along with breakthroughs in the most challenging multi-physics applications, including simulations of the human body or predictions of climate change. A simple question, however, remains: does more compute power automatically lead to more resolved physical effects? My answer to this question is: yes, but this comes with challenges. The *automatically* needs to be dropped.

In 2013, a large group of researchers collected future perspectives of general multi-physics simulations in a common effort [122]. As starting point of my thesis, I want to revisit the inherent complexity of multi-physics simulations as discussed in this publication. Every further added physical effect comes with more parameters and algorithms to tune. Multi-physics complexity does not mean that we can simply add up the complexities of the single-physics simulations. I would say, we have to multiply them if we think about the amount of parameters of every single-physics simulation. Furthermore, the coupling itself brings additional complexity. Thus, we have to develop robust algorithms that work in a variety of situations and that avoid tuning of parameters.

Multi-physics simulations can be distinguished into two overall approaches. The monolithic approach, on the one hand, deduces an overall equation system for a coupled-system, solved at once in a single software. The partitioned approach, on the other hand, couples existing single-physics software on a high level. The monolithic approach can be more robust and more efficient for a single application. If we think, however, about making simulations more realistic by adding or exchanging physical components, only the partitioned approach allows to keep a feasible time-to-solution. By time-to-solution, I refer to the overall time from the developing process of the algorithms and the software to the final simulation. Only reusing existing single-physics experience allows us to tackle the increasing complexity.

This thesis deals with the question how we can translate the black-box flexibility of the partitioned approach into software flexibility while, at the same time, allowing efficient usage of modern, massively parallel computing architectures. As an example, I mainly focus on fluid-structure interaction (FSI). At multiple occasions in this thesis, I add more physical effects to illustrate the increasing complexity. The main goal, however, is not to achieve breakthroughs in FSI itself, but to deduce general algorithms and software concepts that can be applied to any other surface-coupled multi-physics problem.

In the remainder of the introduction, Section 1.1 discusses FSI in more detail and revisits the terms multi-physics and multi-core and their relation. Afterwards, Section 1.2 discusses the two worlds of monolithic and partitioned coupling for FSI. This section comes along with a broad literature review on FSI. Finally, in Section 1.3, I deduce concrete research questions for a scalable, partitioned approach.

1.1 No Multi-Physics without Multi-Core

This section starts with a practical view on FSI in Section 1.1.1. I give a basic definition of FSI and collect various applications. Afterwards, Section 1.1.2, discusses the computational cost of FSI and concludes that high scalability of FSI simulations is a desired goal.

1.1.1 A Practical View on Fluid-Structure Interaction

FSI encompasses all applications in which a solid body is deformed under fluid excitation. In general, the coupling is bi-directional as the deformation of the solid also influences the fluid. This mutual influence is the core of a broad spectrum of practical applications. In the following, I try to list the most important applications. I do not intend to give a full literature overview over single applications, but to give representative publications for every application.

Applications of Fluid-Structure Interaction The first FSI simulations in literature appeared in the middle of the '90s [11, 51, 136, 162, 179, 189, 218, 226] and the importance has ever grown since. One of the most classical application is aeroelasticity – the stability of an elastic body exposed to a fluid flow. This is important for aircrafts [81] or wind turbines [12], but also for lightweight structures such as tents [102, 230]. Other classical engineering applications are parachutes [190, 174] or inflatable structures such as airbags [210]. Marine engineering is another area where FSI plays a prominent role:

fields of research are the interaction of foils – thin profiles, placed under the hull of a ship – with the surrounding water [134] as well as wind interaction of the sail in a sailing boat [159]. In recent years, hemodynamics have played a more and more important role. This includes the simulation of the human heart such as in [120, 167] or the flow in arteries with aneurysms [192, 7]. Exotic FSI applications are the aquatic locomotion of fish [19] or insect flight [194].

Adding Further Physical Effects Many FSI applications are no stand-alone simulations, but are enriched by further physical effects. Heat transfer is a very classical add-on. Often, pure conjugate heat transfer is also referred to as FSI or thermal FSI [21]. In this thesis, however, FSI always refers to the classical mechanical FSI. An important application of added heat transfer are combustion engines [149]. Another example for an added physical effect is acoustics, which is required for noise predictions [130, 175]. Heart simulation requires coupling to electronic propagation [212]. Furthermore, to emulate the full blood circle, coupling to ODEs or 1D models is required. In blood-flow simulation, adding poro-elastic structures can be important [37]. Multiple fluid fields are needed for partially filled tanks on container ships [105], for example. Finally, coupling with control signals is important for many applications [182]. All in all, there is a need for FSI simulations to be inherently flexible and, thus, to allow for an easy extension by further physical effects.

1.1.2 Computational Costs of Fluid-Structure Interaction

In general, FSI is a compute-intensive problem: two possibly non-linear sub-problems need to be solved together. Adding up the costs of both problems, however, underestimates the total cost drastically, as the coupling increases the overall difficulty. At the same time, each field still needs to be solved carefully. Otherwise, the increased modelling effort of moving from single-physics to multi-physics models, does not pay off. For blood flow simulations, for example, moving from a single fluid simulation to an FSI simulation does only give more realistic results if the boundary layer in the fluid domain is carefully resolved. The overall cost increases even more when further physical effects are added. Next, I discuss a concrete turbulent FSI benchmark from literature, which illustrates clearly that flexibility and scalability are both a must, although they are sometimes hard to combine. I use this benchmark to test the scalability of this thesis' concepts in Section 4.5.2. Afterwards, I conclude this section by a literature review on highly scalable FSI approaches.

PfS Benchmark for Turbulent Fluid-Structure Interaction In 2014, De Nayer et al. published a numerical and experimental study of the turbulent FSI benchmarks PfS-1a and PfS-2a [58]. In the same year, they reported on the performance of their numerical approach at the SuperMUC Status and Results Workshop[1] [57]. The benchmark is an important contribution to future FSI research on, for example, light-weight structures, since numerical and modelling approaches can be carefully tested. This includes, for example, the turbulence model, the structural elements, and the FSI coupling approaches. For such scenarios, the flexibility of the FSI approach is essential. The fluid and the structure solver show both very different characteristics, which need to be taken care of by tailored discretization techniques. Specialized single-physics solvers and the related experience need to be exploited in every field to match the experimental data. The authors of the benchmark paper apply a partitioned coupling approach based on the coupling tool CoMA, the predecessor of EMPIRE, which I discuss in Section 2.3. CoMA uses a central serial server, over which all coupling data is communicated and which computes coupling algorithms as well as interpolation methods between non-matching coupling meshes. For the PfS-2a benchmark, the fluid domain is discretized with 13.5 million control volumes to carefully resolve the boundary layer. The structure domain, however, only uses 100 quadrilateral four-node shell elements, which leads to a tremendous cost asymmetry between both domains. Two seconds of physical time are simulated such that enough statistical data is collected for comparison with experimental data. The complete simulation runs on 93 cores, one for the structure solver, one for the coupling server, and all other 91 cores for the fluid solver. This results in a runtime of approximately a thousand compute hours, which is over a month. This is unsatisfactory, in particular because multiple simulation are necessary to find the optimal experimental setup, the correct turbulence model, the correct inflow condition, the right mesh, and so forth. A higher scalability would have a significant influence on the engineer's research.

[1] `https://www.lrz.de/services/compute/supermuc/magazinesbooks/supermuc_results_2014/`

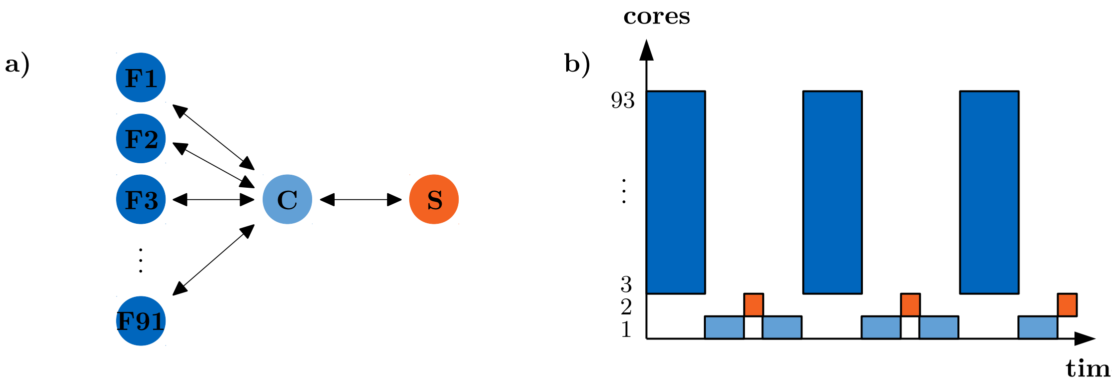


Figure 1: PfS-2a, parallel setup from [57], (a) distribution of tasks to cores, (b) compute and idle periods of the cores over runtime. Scalability is limited due to (a), the central instance, through which all communication runs and which computes the mapping, and (b), the staggered coupling scheme, which probably suffers from the compute intense mapping operation at the central instance. F1 to F91: fluid cores, C: core on which the central instance (coupler) runs, S: structure core.

The fluid solver, at the same time, should allow for a higher scalability. The overall scalability is limited by the coupling approach because the coupling software was developed for smaller scenarios with distinct focus on flexibility, but not on scalability. I already mention above: this thesis' main topic is on how to improve the scalability of such coupling approaches without interfering with the flexibility. Therefore, I solve two problems, which I already want to briefly mention here. Section 1.3 further discusses both problems.

The first problem is the sequential coupling scheme. This means that fluid, structure, and coupling are always computed one after the other. Figure 1 (b), sketches this approach. During the computation of the coupling and during the structural computations, the 91 fluid cores idle. The structural computation might be fast, but the ratio 91 to 1 is also tremendous. The problem becomes more severe if the fluid solver is scaled further.

The second problem is depicted on Figure 1 (a). All coupling computations, including the interpolation between both coupling surface meshes, are executed on the centralized server, which requires communication of data on the complete fluid surface mesh. This 1:N communication limits the scaling of the fluid solver to a higher number of cores. A centralized coupling approach cannot take advantage of the scenario's asymmetry. Furthermore, for the interpolation, a severe compute step at the server is required. Both problems need solutions to further scale up partitioned FSI scenarios. Further research concerning turbulent FSI involves more complex 3D flows, such as the flow around a half-sphere. [227] discusses single-physics fluid results. Here, the fluid mesh even counts 30 million control volumes, so approximately 2.3 times more than for the PfS-2a benchmark. An FSI simulation with the same approach as in [58] is hardly feasible. I present preliminary FSI results for this case using the newly developed concepts of this thesis in Section 5.4.

Scalability of Fluid-Structure Interaction in Literature The scalability limitation of the PfS-2a benchmark is no special case, but a general problem for FSI or, more general, surface-coupled multi-physics problems. Therefore, I end this section with a brief literature review on the most important scalability results for FSI. Partitioned approaches almost always suffer from a centralized server-like instance. To my best knowledge, there is no partitioned FSI simulation published that scales significantly beyond 100 cores. In [121], for example, scalability up to 64 cores is shown. For monolithic FSI, more scalability results are published. Often, the concepts are very complex and specific to single applications. Typically, the challenge is to formulate the right preconditioner for the asymmetric linear system. The higher the asymmetry, the harder it gets to formulate such a preconditioner. For hemodynamics, which marks an only moderately asymmetric case, several scalability studies are available. Cai and several generations of PhD students increased the scalability from 512 cores in [9] (Barker and Cai), to 3,072 cores in [229] (Wu and Cai), up to slightly over 10,000 cores in [125] (Kong and Cai). In different work, several students of Deparis and Quateroni increased the scalability from 768 cores [54] (Crosetto et al.) to 8,096 cores [95] (Forti). Scalability for monolithic simulation is achievable, but comes with many application-specific challenges. The partitioned approach, however, has the advantage that the expertise on how to scale up single-physics can be reused. The solvers Alya and Ateles, which are used in this thesis, for example, have proven to run efficiently on approximately 100,000 cores [213, 239]. Reproducing this scalability in coupled simulations is a natural task. Section 2.4 gives an overview on all single-physics solvers that I use in this thesis.

A highly scalable partitioned approach should have a large impact on engineering. While I already mention this further above for the Pfs benchmarks, this also holds true in general. A faster time-to-solution, including the development time, allows us to develop more complex multi-physics applications, since coupling approaches can be tested and validated earlier. Furthermore, additional algorithmic layers, such as uncertainty quantification or shape optimization become feasible – especially such that have no inherent additional parallel layer. Section 5.6 presents an example for uncertainty quantification.

1.2 On Partitioned and Monolithic Fluid-Structure Interaction

Further above, I already briefly introduce the different philosophies of the monolithic and the partitioned approach for a general multi-physics problem. For FSI, the differentiation between both is sometimes rather fuzzy. Many mixed forms exist. I prefer to look at this as a continuous range of methods as visualized in Figure 2. From left to right, more and more information from single-physics solvers is used, from only black-box access up to full access. Hence, from left to right, a tighter coupling is possible, at the cost of a lower flexibility and a longer software development phase. It is not clear, at which point a method is no longer partitioned, but monolithic. Sometimes, certain mixed-forms are discussed as partitioned in one publication, while similar methods are referred to as monolithic in another publication. In the following, I give an extensive literature review for all sub-steps, organized from left to right. While I try to make this overview exhaustive, this is close to impossible for FSI seeing the vast amount of published methods. While you read these lines, a new method is probably developed. Afterwards, I discuss how the characteristics of the methods in Figure 2 change from left to right and which impact this has.

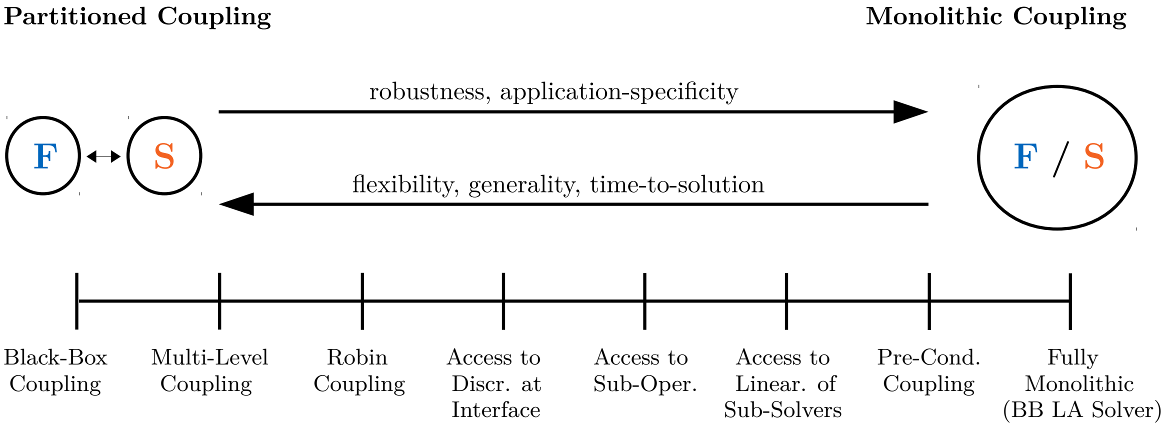

Partitioned Coupling
Monolithic Coupling
robustness, application-specificity
flexibility, generality, time-to-solution
F
S
F / S
Black-Box Coupling
Multi-Level Coupling
Robin Coupling
Access to Discr. at Interface
Access to Sub-Oper.
Access to Linear. of Sub-Solvers
Pre-Cond. Coupling
Fully Monolithic (BB LA Solver)

Figure 2: Overview on various coupling techniques for FSI ranging from minimal black-box access on the left to a fully monolithic approach with a black-box linear algebra solver on the right.

A Whole Range of Methods The partitioned approach can be seen as a domain decomposition method (see e.g. [23]) for the coupled problem: each sub-problem is formulated separately while the coupling is enforced via boundary conditions. While the term *partitioned* is also already introduced in very early publications [87], also different terms are used for the same concept. This applies for the terms *co-simulation* [236] or *staggered* approach [83], for example. I do not prefer the latter expression, as staggered coupling could infer sequential coupling. Partitioned coupling can, however, also be parallel. The complete Chapter 3 deals with such approaches. Black-box methods mark the very left of Figure 2. Such methods only access nodal values at the surface mesh. No discretization details and, therefore, no

Jacobian information is used. Only standard Dirichlet and Neumann boundary conditions are applied. Black-box methods are typically grouped into explicit and implicit schemes. Explicit schemes only perform a fixed amount of solver calls during one timesteps whereas implicit schemes aim to recover the fully coupled solution through sub-iterating. The expressions weakly and strongly coupled are sometimes used similarly. Implicit schemes are necessary to overcome numerical instability caused by a significant added-mass effect. I discuss this issue further down. Purely explicit schemes, often applied for aeroelastic scenarios, are discussed by Farhat et al. [83, 85] or Dettmer et al. [69]. Simple implicit schemes are based on an underrelaxation, possibly via an adaptive Aitken procedure such as discussed in [128]. Quasi-Newton coupling schemes grew in importance during the last 10 years. An earlier work uses a finite-difference approximation of the Jacobian [140]. Jacobian approximations without additional evaluations are discussed in [33, 60, 144, 146, 148, 216]. The schemes that are developed as part of this thesis also fall in this category [129, 142, 202]. A separate literature review in Section 3.3 discusses quasi Newton coupling schemes in detail. Coupling methodologies that operate on a multi-level hierachy of each single-physics domain [29, 65, 178, 209, 240] can still be seen as black-box methods. However, such methods require more effort to establish the coupling. A multi-level hierarchy can be a hierarchy of meshes, but also a hierarchy of models. A combination of multi-level methods with the parallel approaches developed as part of this thesis is discussed in [28]. The next set of methods in Figure 2 are Robin coupling schemes. The stability of the pure Dirichlet-Neumann coupling can be improved by using Robin boundary conditions, which are typically not yet supported by pure single-physics fluid or structure solvers. These methods have been discussed in numerous publications [4, 8, 47, 157], reviewed in [90]. Further methods need access to the discretization of both solvers to assemble and solve an interface system [68, 170, 171]. Even more invasive are methods which add or modify terms in either solver. The coupling stability can be improved by adding an artificial compressibility term in the fluid solver [32, 64]. The authors of [61] show the close relation between this method and Robin boundary conditions, while also discussing limitations of the approach. A similar improvement of stability can be achieved by adding terms in the structure solver [187, 235]. Terms on both sides are adapted in [72]. Semi implicit schemes need access to sub-operators to only iterate pressure terms implicitly while treating the velocity terms explicitly [5, 38, 89]. Besides necessary modification of terms, this results in a further restriction, since the fluid solver must be based on a splitting scheme then. Methods that need access to full linearizations of solvers are sometimes still referred to as partitioned methods [66, 91, 101, 127, 156, 183], but denoted monolithic in other publications such as in [14]. Finally, methods that couple on a preconditioner level can be regarded as truly monolithic. A global Newton linearization leads to a highly ill-conditioned Jacobian. A global preconditioner is constructed by reusing preconditioners of every single-physics domain, possibly in a multi-grid sense. Such schemes are, for example, discussed in [7, 9, 14, 54, 100, 125, 174, 193, 229]. One could argue that a fully monolithic scheme would simply assemble the complete linearized system and use a black-box solver together with a black-box preconditioner. [200] uses such an approach although the Jacobian matrix is not computed explicitly, but approximated. Typically, such schemes significantly suffer from the bad condition of the Jacobian.

Discussion Figure 2 already sketches the most important distinguishing features between a partitioned and a monolithic approach. The partitioned approach allows for a faster code development due to the inherent flexibility and, therefore, the reuse of existing, sophisticated single-physics solvers (see e.g. [12, 22, 46, 83, 86, 102, 140, 175, 194]). The flexibility also allows to easily add further physical components, compare the discussion in Section 1.1.1. Furthermore, tailored numerical schemes can be applied in every domain. A fact that grows in importance if the time and spatial scales in both domains differ significantly. Tailored methods can also be used for pre- and post-processing. On the other hand, the partitioned approach suffers from the inherent added-mass instability [50, 94, 208], caused by freezing the coupling boundary condition during one timestep. The instability increases for relatively light and elastic structures or a fluid with a high viscosity. Contrary, the monolithic approach often requires the development of a new software from scratch. This can also be an advantage as the development can be highly tailored towards a specific application. Furthermore, a monolithic approach allows to derive overall error estimators [168]. Sometimes, people argue that the monolithic approach is better suited for massively parallel simulations. I contradict this believe with this thesis. Also the partitioned approach can be used in a massively parallel setting. In a way, even less communication is necessary, since values only need to be exchanged after every coupling iteration. The locality of the computation, meaning that neighboring domains are also solved on neighboring processors, however, might be better for a monolithic approach, based on an overall mesh decomposition. Time adaptivity and higher temporal order is possible for both approaches – the partitioned approach [22, 211] and the monolithic approach [53, 141]. The FSI literature provides several comparisons between both approaches. An early work is [145], which concludes on the higher stability of the monolithic approach by means of a 1D example. Several publications aim for runtime comparisons (e.g. [60, 111, 180]), in which the monolithic approach typically slightly outperforms the partitioned approach. A fair runtime comparison is, however, not only close to impossible, but, in my opinion, also the wrong angle. The goal of the partitioned approach is not to compete runtime-wise, though this is sometimes possible, but to provide a faster development time. Often, the choice upon the better approach is dominated by the application. FSI in hemodynamics features a high added-mass effect, but similar time and spatial scales. A monolithic approach might, therefore, be advantageous. Aeroelastic applications, on the other hand, have smaller added mass effects, but different scales. Here, the partitioned approach is normally used. I give the main reason to intensify the research into the partitioned approach at the very beginning of this thesis. Multi-physics application become more and more complex and need, therefore, the inherent flexibility of the partitioned approach. Only then, a feasible time-to-solution is possible. Prototyping new applications must be easy. Software and robust algorithms that can cover multiple applications should be our research guideline.

Treatment of Moving Geometries As a final remark of this section, I collect information on the treatment of moving geometries in the fluid solver. This is also an important characterization of FSI approaches, but plays no central role in this thesis. Still, for the sake of completeness, I want to mention several aspects. Two basic methods exist to resolve moving geometries. Either the reference frame of the fluid solver and therewith also the mesh is moved with the geometry, or the boundary condition on the geometry is enforced in a weak sense. For the fluid solver, the first approach uses an arbitrary-Lagrangian-Eulerian (ALE) framework, while the second approach uses a pure Eulerian approach. Numerous expressions for both approaches exist: conforming versus non-conforming, moving grid versus fixed-grid, interface tracking versus interface capturing, and so forth. Often methods of the second group are also called immersed boundary methods. Methods of the first group, the ALE group, are typically better suited to resolve boundary layers, but require re-meshing techniques for too large movements and fail to handle topology changes in the geometry. The second group, the immersed boundary group, can handle arbitrarily large movements and topology changes but mesh elements are not aligned with the geometry. Both approaches can be used with either a structured or an un-structured mesh. Immersed boundary methods for FSI have first been considered by Peskin [160] in 1972. A good, but already slightly outdated review can be found in [150]. In recent years, methods based on the Nitsche approach [155] got very popular in FSI (see e.g. [16]). Besides the two main groups, also more exotic approaches exist, such as a combination of ALE and the immersed boundary method in [82], complete Eulerian FSI [74, 75, 221] or complete Lagrangian FSI [117]. The most important conclusion from this paragraph is that this characterization of FSI can be regarded fully independent from the equation coupling.

1.3 Challenges of a Scalable Partitioned Approach

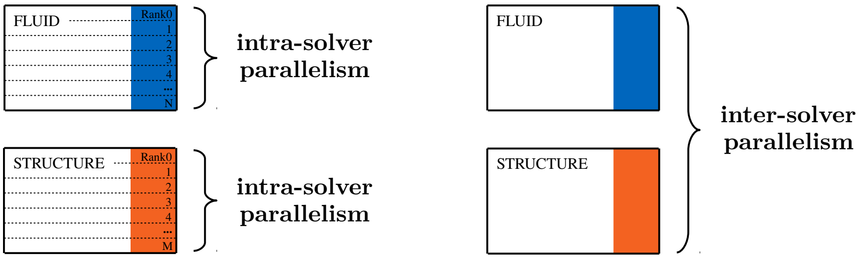


Figure 3: Two layers of parallelism for an FSI simulation: intra-solver parallelism, left, see Chapter 4, and inter-solver parallelism, right, see Chapter 3.

Further above, I mention two problems which prevent the PfS benchmarks from reaching a higher scalability: idling processors due to a sequential coupling scheme and a central instance through which all coupling meshes need to be sent. In this section, I deduce two challenges from theses two problems. Both can be seen as introducing parallelism on different levels. First, an inter-solver parallelism, meaning a coupling scheme that allows the simultaneous execution of both solvers. Second, an intra-solver parallelism, meaning the computation of all interface numerics directly on the solver ranks and not on a central instance. Figure 3 sketches both parallelization levels. These two parallelization levels are the two main topics of this thesis. While the first one relies on the mathematical, numerical branch of scientific computing, the second one is rather a computer science, high-performance computing topic. Next, I present both challenges and briefly sketch their solution: the inter-solver parallelism in Section 1.3.1 and the intra-solver parallelism in Section 1.3.2.

1.3.1 Avoiding Idling Processes by Parallel Coupling Schemes

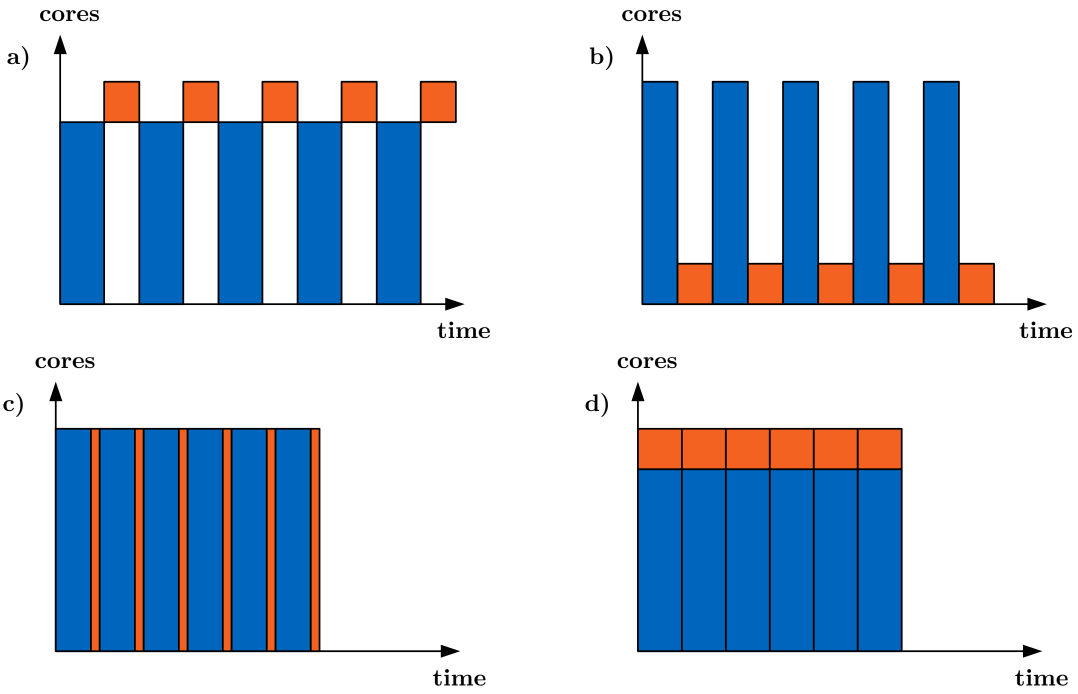


Figure 4: Parallel layout of different FSI coupling strategies. Blue and orange rectangles refer to fluid and structure computations, respectively. a) Classical sequential schemes let idle half of the processes in average. b) Reusing compute resource does only result in a slightly better situation. c) Unfeasible layout: the structure solver does not scale on all fluid cores due to the immense cost asymmetry. d) A parallel coupling leads to a clean solution.

Classical partitioned coupling schemes for FSI use a sequential coupling of the fluid and the structure solver. They are solved one after the other. Figure 4 (a) depicts this situation. Half of the processes idle in average. Theoretically, one could reuse the processes of each solver for the other (discussed e.g. in [49]). Technically, this is far from trivial. Still, it only leads to the situation depicted in Figure 4 (b) and not to the situation of Figure 4 (c). Most FSI scenarios, such as the earlier discussed PfS benchmarks, feature an immense asymmetry between both solvers. The fluid problem consists of many degrees of freedom, which makes it compute intensive. At the same time, this makes the fluid problem also scalable on many cores. The structure problem, on the other hand, is cheap and scales, thus, only on a few cores. The situation of Figure 4 (c) is not feasible since the structure solver does not scale on the same amount of cores as the fluid solver. In the limit case, going from (a) to (b) results in no speed-up. This drawback of the sequential coupling worsens if more than two solvers are coupled. For n coupled solvers, at worst, a ratio of $(n-1)/n$ of all processes constantly idle. The only clean solution to this problem is a parallel coupling scheme, allowing the simultaneous execution of the fluid and the structure solver. In case of a perfect load-balancing, this results in no idling processes as depicted in Figure 4 (d). Compared to (a), a theoretical speed-up of two is possible. For explicit coupling, such parallel coupling schemes have been known and used since the very beginning of FSI simulations [83, 86]. For implicit coupling, the story is a different one. If the number of iterations to converge to a fixed convergence criterion doubles when going from the sequential coupling (a) to the parallel coupling (d), nothing is won. For non-black-box coupling schemes, implicit parallel approaches already exist [66, 170]. One of the main contribution of this thesis is the development of implicit parallel black-box coupling schemes that feature the same convergence speed as their sequential, state-of-the-art counterparts. The construction of such parallel coupling schemes is not overly complicated. The state-of-the-art sequential coupling scheme IQN-ILS [60] is a combination of the sequential Gauss-Seidel coupling and an Anderson acceleration [1]. The sequential part relies on the sequential non-linear block-Gauss-Seidel fixed-point equation. Replacing this sequential fixed-point equation by a parallel one, such as a block-Jacobi one, leads to a parallel coupling scheme. Balancing both components via a dynamic weighting restores the same convergence speed. Chapter 3 discusses all these topics in detail. An interesting work about the inverse direction has recently been presented in [131]. Here, the parallel Jacobi like solver ASPIN has been transformed into its sequential counterpart MSPIN, to improve the robustness and the convergence speed of the algorithm. This does not hold true for the methods developed in this thesis: the robustness and convergence speed of the parallel coupling schemes is similar to the sequential ones.

1.3.2 Avoiding a Central Coupling Instance by a Pure Peer-To-Peer Approach

Many coupling tools use a central instance to manage the overall workflow. This includes the earlier discussed results for the PfS benchmarks. The central instance acts as a server and is often executed in a single thread. Each rank of a parallel single-physics solver registers as a client at the server. All communication between different solvers runs through the central server. Coupling data is held at the server. Accessing this data requires communication. All interface numerics such as interpolation between non-matching grids or coupling schemes are executed on the central instance. This setup is sketched in Figure 5 (a). It is not astonishing that most early developments of multi-physics coupling software chose this approach. The central instance can easily overview the overall steering logic. 10 years ago, it was important to simplify things since many other challenges such as finding stable coupling schemes and the right interpolation methods were still ahead. For massively parallel simulation, this centralized layout is however not suitable for different reasons:

1. The 1:N communication between each solver and the server can result in a throughput bottleneck.
2. Many small messages are required for accessing coupling data if the solver features a de-centralized data structure.
3. All coupling meshes and the associated coupling data need to be communicated to the server.
4. All interface numerics are computed in serial.

For a certain application, it is not always trivial to judge upon the most severe bottleneck among these four. One should be careful with too early conclusions. In [99], a parallelization of the server is proposed. A similar approach is used in [121]. Such a parallel solver could be a remedy for problem 1 and 4, and also partially for problem 2, but not for problem 3.

A clean solution for all four problems is a fully parallel peer-to-peer approach as depicted in Figure 5 (b). No central instance needs to be executed. Communication of coupling data between solvers works locally. Interface numerics are directly computed on the solver ranks on distributed data. For highly asymmetric cases, only the smaller coupling mesh has to be communicated. Such a pure parallel peer to peer approach is realized as part of this thesis. The starting point of this realization is the coupling library preCICE. Next, I first briefly introduce preCICE before then arguing why preCICE marks the perfect starting point for the development of a parallel peer-to-peer coupling concept.

The Coupling Library preCICE preCICE is the successor of the coupling tool FSI∗ce. The latter was developed by Markus Brenk [34] more than 10 years ago as part of the *DFG Research Unit 493 Fluid-Structure Interaction: Modelling, Simulation, Optimization*[2]. FSI∗ce was a server-based coupling

[2] http://fsw.informatik.tu-muenchen.de/

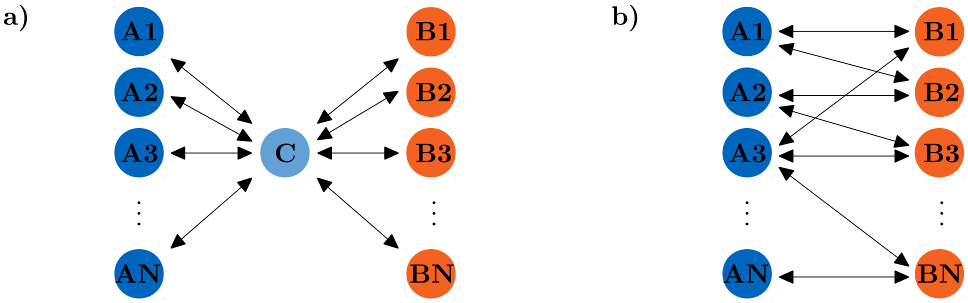


Figure 5: Parallel layout of a coupling approach using a central instance (a) compared to parallel peer-to-peer approach (b).

tool. Bernhard Gatzhammer continued this development by re-developing the server-based concept of FSI∗ce into a peer-to-peer concept for preCICE [99]. This peer-to-peer concept, however, still uses a server per solver. This means, for two coupled solvers, two server threads need to be executed. My thesis renders central instances completely unnecessary by introducing a fully parallel concept. preCICE offers various coupling schemes, methods for interpolation between non-matching coupling meshes and means for communication between separate executables. The library features a high-level application programming interface (API), which makes it easy to use. Chapter 2 introduces preCICE thoroughly along with a review of alternative coupling software.

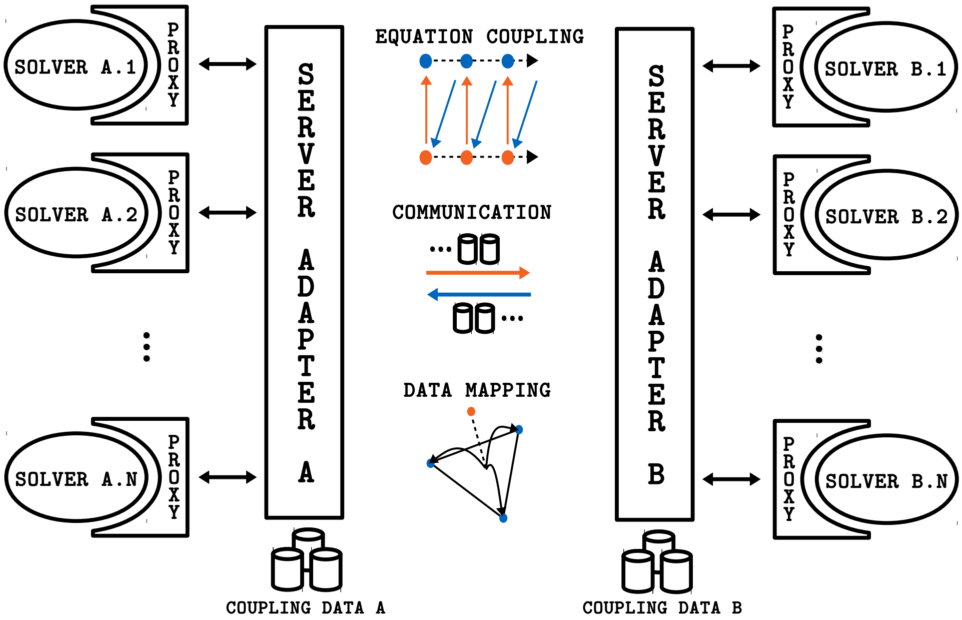


Figure 6: Old parallelization concept of preCICE [99]. Coupling data is held at a server for each parallel solver. Accessing data requires communication to the server. Interface numerics are executed on the servers. A single communication channel is established between both servers.

Porting preCICE to a Fully Peer-to-Peer Layout Figure 6 sketches the old parallelization concept of preCICE. Whereas it already features a peer-to-peer layout – no central instance is present – each solver still needs a separate server thread. The coupling data is held at this server thread and all interface numerics are computed there. For the interpolation, the coupling mesh of one solver is communicated from its server to the other. The old parallelization concept still suffers from all four problems that I discuss further above. The steering logic, however, works already without any central instance. Porting this layout to fully peer-to-peer concept without any servers is, thus, much easier than starting from a central-server based tool. The high-level API, on the other hand, provides the full flexibility of the partitioned approach as discussed at the very beginning of this thesis. Furthermore, ready-to-use adapters for sophisticated single-physics solvers are already available. Also, preCICE is a long-term software project with a well-designed software architecture, a full unit-testing of every single component and various integration tests of the API. All in all, the original preCICE, as in [99], marks the perfect starting point for developing a coupling tool for massively parallel multi-physics simulations.

Before briefly describing the new parallelization concept of preCICE, I want to discuss a first introductory experiment. I perform a strong scaling study of the Ateles Cube scenario with the original server-based parallelization concept. The Ateles Cube scenario simulates a density pulse travelling through a cubical Euler domain. For testing purpose, the cube is cut at an artificial interface and coupled via preCICE. Section 4.5.1 gives a detailed explanation of the testcase and the experimental setting. Figure 7 shows the time per timestep of the coupled simulation compared to a monolithic simulation. The coupled simulation shows a drastic overhead. The overhead is due to the large amount of small messages that each solver rank has to send to access data. From 32 to 64 and from 256 to 512 total cores, this overhead remains constant as the amount of cores at the interface does not increase, compare Section 4.5.1. With the new parallelization concept of this work, the coupled simulation features almost no overhead compared to the monolithic simulation. The impatient reader might directly want to peek at Figure 79 on page 100. Furthermore, no scalability degeneration for up to 32,000 cores is visible [181].

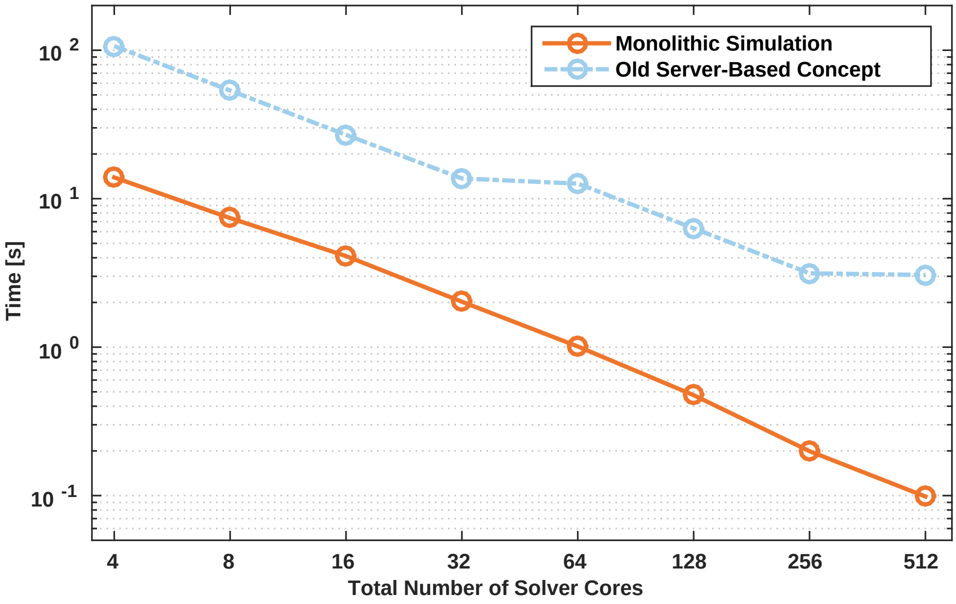


Figure 7: Strong scaling of the work per timestep for the Ateles Cube. A monolithic simulation is compared to the old server-based parallelization concept of preCICE. The same figure, complemented by the results of the new parallelization concept is used in Section 4.5.1, Figure 79. For the old server-based concept, the resources for both server processes are neglected. The initialization of the server-based coupling takes approximately 50s independent of the total number of cores. This is mainly due to the serial compute effort of the nearest-neighbor mapping.

Figure 8 sketches the new parallelization concept of preCICE. Data is stored locally at each solver rank. All three feature groups of preCICE – equation coupling, interpolation, and communication – are executed at the solver ranks on distributed data. At the same time, the API of preCICE and therewith the flexibility is not changed at all. Scalability is combined with flexibility – the core of this thesis. The usability of preCICE even improves as no server threads need to be started any longer. With the original preCICE, pinning these threads is a non-trivial task. The basis for the parallelization is the re-partitioning of communicated coupling meshes at the receiver. This re-partitioning needs to be computed in a way such that the interpolation methods can be computed locally. Afterwards, local communication channels between both solvers can be established. A general assumption of this thesis is that the re-partition remains constant throughout a simulation. This is true for many applications such as coupling at bounding boxes or FSI between a Lagrangian structure solver and an arbitrary-Lagrangian-Eulerian fluid solver. FSI with an Eulerian fluid solver would contradict this assumption. Still, building blocks from this thesis can be re-used for this future challenge. Due to this assumption, I do not port the geometry interface of preCICE (compare [99]) to the new parallel layout. This is out of scope for this thesis, though I expect no fundamental limitations. A further conclusion from the constant re-partition is the different importance of the initialization phase compared to the work per timestep. The initialization, including the re-partitioning of the received coupling mesh is only executed once per simulation, compared to a possibly tremendous amount of timesteps and coupling iterations. For the earlier discussed Pfs-2a benchmark, for example, [58] uses $2 \cdot 10^5$ timesteps. Thus, the main focus of the new parallelization concept of this thesis is to optimize the time per timestep which is spent in preCICE while achieving a tolerable initialization time. As tolerable, I consider an initialization time for the coupling that is of the same order as the initialization effort of each single-physics solver. For the largest cases with 10^5 cores and 10^9 total unknowns, an initialization time below 10s is the goal. As the coupling interface is a lower-dimensional manifold of the complete simulation domain, this goal is achievable by simple global operations. The work per timestep should, however, not include any unnecessary global operations and, thus, be ready for the forthcoming exa-scale machines. The new parallelization concept of preCICE is studied in detail in Chapter 4.

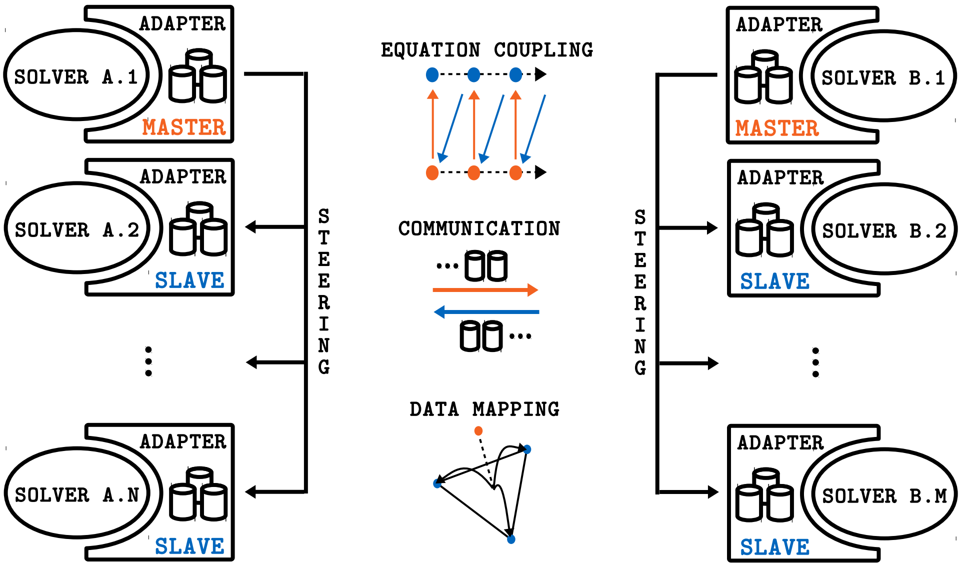


Figure 8: New parallelization concept of preCICE, developed as part of this thesis. Coupling data is stored locally at each solver rank. Interface numerics are, thus, executed at the solver ranks on distributed data. Communication is fully local between individual solver ranks.

Application of both Parallel Levels The two new parallelization levels – the inter-solver parallelism in Chapter 3 and the intra-solver parallelism in Chapter 4 – can be studied independently of each other. The only reason to read Chapter 3 before Chapter 4 is the realization of the coupling schemes on distributed data, discussed in Section 4.4 which builds upon the serial description in Section 3.2. Both chapters need the introduction to preCICE in Chapter 2 as a prerequisite. Afterwards, in Chapter 5 various show cases, encompassing applications in aeroelasticity and hemodynamics, demonstrate the full potential if both parallel layers are combined.

About this Thesis I expect the reader of this thesis to have a basic understanding of computational fluid dynamics and computational structural mechanics. To fill possible knowledge gaps, I recommend [14, 106, 164, 223, 228]. The thesis of Bernhard Gatzhammer also gives a nice introduction [99]. Multi-physics simulations, in particular in a high performance computing context, are always a team effort. Therefore, it is quite obvious that parts of my thesis are the product of joint work. I mention my collaborators at the beginning of each chapter separately.

Summary of Chapter 1

- To advance multi-physics research, in particular at the dawn of exascale, a flexible and scalable simulation environment is necessary.
- A partitioned black-box coupling strategy gives the required flexibility. This thesis deals with the question on how to make such a strategy scalable.
- Therefore, FSI is studied as an important and challenging representative of a general multi-physics problem. The goal, however, is to derive general concepts.
- FSI itself has a tremendous need for more scalability, in particular partitioned FSI. To obtain more scalability, parallelism on two levels is studied in this thesis.
- Inter-solver parallelism: parallel coupling schemes allow for a simultaneous execution of multiple solvers. I study such schemes in Chapter 3.
- Intra-solver parallelism: a coupling tool without central instances avoids coupling bottlenecks. I study such a concept in Chapter 4.
- The coupling library preCICE marks a perfect starting point for theses developments, since preCICE features a high flexibility and already uses a peer-to-peer layout. In Chapter 2, I give an introduction to preCICE.
- Finally, in Chapter 5, I study how the two new parallel layers can be used for practical applications.

2 An Introduction to preCICE

In the introduction to this thesis, in Section 1.3.2, I motivate why the coupling library preCICE marks a perfect starting point for this thesis. preCICE is a coupling library that uses a peer-to-peer approach and features a high-level application programming interface (API). preCICE offers numerical methods for equation coupling, methods to interpolate between non-matching meshes, and means for communication between separate executables. This chapter gives a compact introduction to preCICE. Many aspects of this chapter are already described in [99] and are repeated here mainly for the sake of completeness and as a starting point for the explanations in the next two chapters. One of the strengths of this thesis is that all necessary adaptions allowing for an executions of preCICE on distributed data and allowing for a simultaneous execution of multiple solvers, do not result in any alteration of the API of preCICE. Thus, the high level of flexibility of preCICE is not reduced. Furthermore, solver adapters do not need to undergo any changes. Still, backward compatibility is achieved, such that the old server-based parallelization concept (cf. Figure 6) is still fully functional. I focus my explanations in this chapter, in the rare cases where necessary, however, on the new fully parallel concept (cf. Figure 8).

preCICE is a joint software project, currently developed by Florian Lindner, Klaudius Scheufele and myself. As already mentioned in the introduction, the original software was implemented by Bernhard Gatzhammer [99, 39], based on earlier work by Markus Brenk [34]. The software is open-source and available on `github`[3]. We published the recent state of preCICE in [42]. In this chapter, I introduce preCICE in two steps. First, Section 2.1 gives a detailed explanation of the user perspective of preCICE, meaning everything a user of preCICE needs to know in order to couple his or her solver to any other solver. This includes the API of preCICE, a list of features as well as an overview of the configuration. I focus on a driving code example to give the reader a simple first impression. Second, Section 2.2 introduces the developer perspective, meaning everything a developer of preCICE needs to know in order to implement new features. This part acts mainly as a starting point for the new implementations presented in Chapters 3 and 4. Afterwards, the chapter closes with a review of alternative coupling software in Section 2.3 and an introduction of all coupled single physics solvers used in this thesis in Section 2.4.

2.1 User Perspective

Building preCICE gives the user the library `libprecice` and an auxiliary executable `binprecice`. The latter can be used to start a server for the old parallelization concept, to run tests, or to auto-generate an xml reference. In this section, I first introduce the reader to the API of preCICE, i.e., I explain how to incorporate `libprecice` into a single-physics solver. Afterwards, I have a closer look at the features of preCICE as well as at the configuration.

2.1.1 Application Programming Interface

preCICE is written in `C++`. The API is, however, available in all major scientific programming languages: `C++`, plain `C`, `Fortran 90/95`, `Fortran2003` and `Python`. The code listings in this section are restricted to `C++`, though.

The API is located in the class `SolverInterface`, which is enclosed in the namespace `precice`. Figure 9 lists the constructor of the class along with the API function `configure`. As input arguments, the constructor needs the name of the solver and rank and size of the current thread for parallel runs. The solver thread with rank 0 internally serves as a master thread to preCICE, as detailed in Section 4.1. The API function `configure` reads and validates an xml file to configure all coupling features at run-time. Afterwards, the master-slave communication, i.e. the intra-solver communication, is set up. Section 2.1.3 gives more details on the configuration. For procedural programming languages such as `Fortran`, both functions are combined into one routine. For the sake of simplicity, I omit in the following the prefix `SolverInterface` as well as the namespace `precice`. Next, I explain the rest of the API functions in three groups: steering methods, mesh and data access and auxiliary methods. I do not cover the geometry API as it is not part of this thesis and has not yet been ported to distributed data. Still, it is fully functional in the server mode and explained in detail in [99].

[3] https://github.com/precice/precice

```
namespace precice {
    class SolverInterface
    {
        public:
        SolverInterface (
            const std::string& solverName,
            int                solverProcessIndex,
            int                solverProcessSize );

        void configure (
            const std::string& configurationFileName );
    }
}
```

Figure 9: preCICE API: main class `SolverInterface`. The constructor uses the solver name as well as the rank and size of the current thread as input arguments. `configure` allows for a configuration at run-time by means of an xml file.

I use a driving example for all API functionality to give the reader a simple first impression of preCICE. Figure 10 lists the original single-physics state of the example solver. This example solver plays the role of a fluid solver in a fluid-structure interaction scenario and consists of some initialization and finalization calls as well as a simple time loop. I refer to the solver as `FluidSolver`. In preCICE nomenclature a solver is also refered to as a participant of a coupling. This example is already used in a couple of preCICE publications (e.g. [42]) as well as on the preCICE `github` wiki.

```
turnOnSolver(); //e.g. setup and partition mesh
while (not simulationDone()){   // time loop
  beginTimeStep(); // e.g. compute adaptive dt
  computeTimeStep();
  endTimeStep(); // e.g. update variables, increment time
}
turnOffSolver();
```

Figure 10: Driving API example without preCICE functionality. The example `FluidSolver` consists of an initialization and a finalization call as well as a simple time loop.

Steering Methods The steering API methods allow the user to steer the behavior of preCICE. This group consists of the four methods `initialize`, `initializeData`, `advance`, and `finalize`, which are listed in Figure 11.

```
double initialize();
void initializeData();
double advance ( double computedTimestepLength );
void finalize();
```

Figure 11: preCICE API: steering methods. `initialize` sets up data structures and communication channels, `initializeData` can be used optionally to communicate non-zero initial data to other solvers, `advance` indicates preCICE to advance the overall coupling procedure after each timestep or coupling iteration and `finalize` tears down data structures and closes communication channels.

`initialize` initializes the coupling. To do so, various steps are performed. First, data structures and master communication channels to other participants are set up. Next, meshes are communicated and, if necessary, re-partitioned (cf. Section 4.1). Afterwards, communication channels between slaves are established (cf. Section 4.2). Optionally, `initialize` can also cover the computation of static mappings (cf. Section 4.3), the reading of restart data, or the first receiving of coupling data in a serial coupling scheme (cf. Section 3.5). `initialize` returns the maximum timestep length the solver should compute next. This functionality can be used to impose the timestep length by one coupling participant onto another. Afterwards, `initializeData` can be used optionally to communicate non-zero initial data to other participants. This can be helpful, for example, if a structure solver starts in a non-referential state after a restart or if an acoustic solver needs a physical condition close to a background state for a successful first timestep. If no initial data communication is configured, this method returns without any effect.

`advance` needs to be called after the computation of every timestep to indicate this to preCICE. Here, preCICE applies mappings schemes, communicates coupling data, and computes fixed-point acceleration techniques. Furthermore, exports to track certain so-called watchpoints, to visualize coupling data or to write restart data, are treated if required. As an argument, the last timestep size has to be passed to `advance` to inform preCICE about a possible subcycling. The return values indicates, again, the next maximum timestep size. Finally, `finalize` tears down data structures and closes communication channels. In Figure 12, the driving code example is extended by the steering methods.

```
turnOnSolver(); //e.g. setup and partition mesh
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

double dt; // solver timestep size
double precice_dt; // maximum precice timestep size

precice_dt = precice.initialize()
while (not simulationDone()){  // time loop
  beginTimeStep(); // e.g. compute adaptive dt
  dt = min(preciceMaxDt, dt);
  computeTimeStep();
  precice_dt = precice.advance(dt);
  endTimeStep(); // e.g. update variables, increment time
}
precice.finalize();
turnOffSolver();
```

Figure 12: Driving API example, extented with the steering API.

Mesh and Data Access To couple two participants at a common coupling interface, both need to define a surface mesh at this interface, the so-called coupling mesh. Coupling meshes and their data fields are defined in the configuration (cf. Section 2.1.3). These data structures can now be accessed via the API. Therefore, each mesh or data entry and also each mesh vertex is identified via an integer ID. Figure 13 lists all necessary mesh API methods. `hasMesh` allows to check if a certain mesh is defined in the configuration, while `getMeshID` returns the ID for a certain mesh. To define the actual vertex coordinates, `setMeshVertex` allows to define a single mesh vertex and returns a vertex ID to reference this vertex. For performance reasons, multiple vertices can be defined at once via `setMeshVertices`. Afterwards, optional connectivity information can be added via `setMeshEdge` and `setMeshTriangle`. Such connectivity information is only needed for projection mappings or geometry queries.

Once the mesh data structure is defined, coupling data can be accessed. Figure 14 collects the relevant API methods. preCICE distinguishes between scalar and vector valued data. `writeVectorData`, for example, allows to write vector-valued data to the preCICE data structure. For performance reasons, again, multiple data values can be written at once via `writeBlockVectorData`. Similar methods exist for scalar data and for reading data. Figure 15 shows the driving API examples, extended by mesh and data access. The code is ready to use for explicit coupling. Still, small modifications need to be applied for implicit coupling, which I explain next.

Auxiliary Methods preCICE offers various auxiliary methods for specific solver needs. For example, actions allow to trigger certain events. Figure 16 lists the corresponding API methods. Via

```
bool hasMesh ( const std::string& meshName ) const;
int getMeshID ( const std::string& meshName );

int setMeshVertex (
    int           meshID,
    const double* position );

void setMeshVertices (
    int     meshID,
    int     size,
    double* positions,
    int*    ids );

int setMeshEdge (
    int meshID,
    int firstVertexID,
    int secondVertexID );

void setMeshTriangle (
    int meshID,
    int firstEdgeID,
    int secondEdgeID,
    int thirdEdgeID );
```

Figure 13: preCICE API: coupling mesh access. `setMeshVertex` and `setMeshVertices` allow to define vertex coordinates for a certain mesh. With `setMeshEdge` and `setMeshTriangle` optional connectivity information can be added.

```
bool hasData ( const std::string& dataName, int meshID ) const;
int getDataID ( const std::string& dataName, int meshID );

void writeVectorData (
    int           dataID,
    int           vertexID,
    const double* value );

void writeBlockVectorData (
    int     dataID,
    int     size,
    int*    vertexIDs,
    double* values );
```

Figure 14: preCICE API: data access. `writeVectorData` and `writeBlockVectorData` allow to write vector-valued coupling data into the preCICE data structures. Similar methods exist for scalar data as well as for reading data.

`isActionRequired`, the necessity of a certain action can be inquired. With `fulfilledAction`, on the other hand, the user informs preCICE once a certain action has been fulfilled. Actions are referenced via strings, which are defined in the nested namespace `constants`. `actionReadIterationCheckpoint` and `actionWriteIterationCheckpoint` allow, for example, to steer an implicit solver coupling. Further auxiliary methods are collected in Figure 17. `isTimeStepCompleted` and `isCouplingOngoing` also allow to trigger specific actions in the solver, or the end of the simulation, respectively. `isReadDataAvailable` and `isWriteDataAvailable` can prevent unnecessary data access in case of subcycling. Figure 18 lists the necessary changes for the driving example to enable the capability for implicit coupling.

```
turnOnSolver(); //e.g. setup and partition mesh
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");

int dim = precice.getDimension();
int meshID = precice.getMeshID("FluidMesh");
int vertexSize; // number of vertices at wet surface
// determine vertexSize
double* coords = new double[vertexSize]; // coords of vertices at wet surface
// determine coordinates
int* vertexIDs = new int[vertexSize];
precice.setMeshVertices(meshID, vertexSize, coords, vertexIDs);
delete[] coords;

int displID = precice.getDataID("Displacements", meshID);
int forceID = precice.getDataID("Forces", meshID);
double* forces = new double[vertexSize*dim];
double* displacements = new double[vertexSize*dim];

double dt; // solver timesetp size
double precice_dt; // maximum precice timestep size

precice_dt = precice.initialize()
while (not simulationDone()){  // time loop
  beginTimeStep(); // e.g. compute adaptive dt
  dt = min(preciceMaxDt, dt);
  computeTimeStep();
  computeForces(forces);
  precice.writeBlockVectorData(forceID, vertexSize, vertexIDs, forces);
  precice_dt = precice.advance(dt);
  precice.readBlockVectorData(displID, vertexSize, vertexIDs, displacements);
  setDisplacements(displacements);
  endTimeStep(); // e.g. update variables, increment time
}
precice.finalize();
delete[] vertexIDs, forces, displacements;
turnOffSolver();
```

Figure 15: Driving API example, extended by mesh and data access. For convenience, the solver functions `computeForces` and `setDisplacement` are added. The solver is ready to use for explicit coupling.

2.1.2 Features

In Section 1.3, I mention the three main feature groups of preCICE: coupling schemes, communication, and interpolation methods (cf. Figure 8). In this section, I briefly list all available options for these three feature groups as they are available after this thesis. I give a brief literature review of mapping methods along with the list for those methods that are implemented in preCICE. For coupling schemes, Section 3.3 already gives a broad literature review. I, thus, avoid the repetition here.

Interpolation Methods For the sake of generality, preCICE only considers fully non-conforming meshes. This means that not even mesh elements of two surface meshes are aligned to each other, but that there can be gaps and overlaps. As preCICE is built for black-box coupling (cf. Section 1.2), mapping methods cannot access shape functions of any of the solvers. preCICE offers two kinds of mapping methods: projection-based mapping methods and radial basis function (RBF) interpolation. Both types are supported in a consistent variant, guaranteeing the exact mapping of constant values,

```
bool isActionRequired ( const std::string& action );
void fulfilledAction ( const std::string& action );

namespace precice {
    namespace constants {
        const std::string& actionWriteInitialData();
        const std::string& actionWriteSimulationCheckpoint();
        const std::string& actionReadSimulationCheckpoint();
        const std::string& actionWriteIterationCheckpoint();
        const std::string& actionReadIterationCheckpoint();
    }
}
```

Figure 16: preCICE API: action methods. `isActionRequired` can trigger specific events in the solver. On the other hand, via `fulfilledAction`, the solver can tell preCICE about the successfull fulfillment of such an action. Actions are referenced via strings and various possibilities are defined in the nested namespace `constants`.

```
bool isTimestepComplete();
bool isCouplingOngoing();
bool isReadDataAvailable();
bool isWriteDataRequired ( double computedTimestepLength );
```

Figure 17: preCICE API: further auxiliary methods. `isTimeStepComplete` and `isCouplingOngoing` allows to steer specific events in the solver. `isReadDataAvailable` and `isWriteDataAvailable` can prevent unnecessary data access in case of subcycling.

and in a conservative variant, guaranteeing the conservation of integral values. Section 4.3.1 gives formal mathematical definitions of these two terms. Projection-based mapping methods comprise a nearest-neighbor mapping and a nearest projection mapping. While the first method is a first order scheme, the latter one is second order if the projection distance from one mesh to the other is much smaller than the mesh width. In practice, this typically holds. [34] gives more information. RBF mappings use a linear combination of vertex-centered radial-symmetric basis functions, together with a single globally defined polynomial. preCICE supports various such basis functions, which can be grouped by their support into global and local functions and lead, therefore, to dense or sparse system matrices, respectively. The original server-based preCICE version supported already a serial LU decomposition to solve the corresponding system matrix. This is, however, highly inefficient for sparse matrices and not well-suited on distributed data. This thesis introduces an iterative solver procedure based on the PETSc library [6]. Section 4.3.2 gives details on the realization.

The thesis of Bernhard Gatzhammer already gives a detailed literature review on mapping methods [99]. I give a summary of the most important contributions and an update on recent developments. The first mapping approaches for FSI were projection mappings [51, 136]. Farhat et al. introduced the concept of a conservative approach [84] to better handle conservation-critical data values. [31] got known as a standard review paper, which also details the unphysical oscillations a conservative mapping can cause. In recent years, mortar methods [18] became the most popular choice if access to the shape functions is given. These methods use a Galerkin approach at the interface, which leads to a simple mass-matrix system to solve. Part of the popularity stems from the fact that an elegant combination with isogeometrical analysis is possible [13]. Furthermore, using dual test functions, the mass matrix system can degenerate to a plain diagonal system. This approach got known as dual mortar method [225] and is used for FSI, for example, in [124]. [88] is a further valuable review paper and [220] a very recent work, which compares numerical results for mortar, dual mortar and projection methods.

RBF interpolation is a very powerful approach if only pure scattered data, without any mesh connectivity information, is given. The most important early works are [36, 185]. Beckert et al. first applied RBF methods in FSI [15]. While RBF methods with global support give better convergence properties, they suffer under an inherently bad condition, in particular for very large systems [36]. Recent work tries to

```
turnOnSolver(); //e.g. setup and partition mesh
precice::SolverInterface precice("FluidSolver",rank,size);
precice.configure("precice-config.xml");
const std::string& coric = precice::constants::actionReadIterationCheckpoint();
const std::string& cowic = precice::constants::actionWriteIterationCheckpoint();
[...]
precice_dt = precice.initialize()
while (precice.isCouplingOngoing()){
  if(precice.isActionRequired(cowic)){
    saveOldState(); // save checkpoint
    precice.fulfilledAction(cowic)
  }
  [...]
  precice_dt = precice.advance(dt);
  [...]
  if(precice.isActionRequired(coric)){ // timestep not converged
    reloadOldState(); // set variables back to checkpoint
    precice.fulfilledAction(coric)
  }
  else{ // timestep converged
    endTimeStep(); // e.g. update variables, increment time
  }
}
precice.finalize();
[...]
turnOffSolver();
```

Figure 18: Driving API example, adaptions to enable the capability for implicit coupling. This is realized via actions for reading and writing iteration checkpoints. The solver then needs to provide the methods `saveOldState` and `reloadOldState`. `isCouplingOngoing` lets preCICE steer the end of the simulation run.

overcome this drawback and make RBF methods ready for massively parallel applications. Torres et al. cut the support from global to local, but recover global influence by iterating [195]. [234] discusses already a PETSc-based realization. [67] uses a local rescaling to avoid the classical polynomial term and therefore allow for an easier and more efficient parallelization. RBF methods are also a popular approach for many other applications besides data mapping. A full review of such work is beyond the scope of this thesis. Important for FSI is, for example, also a mesh moving technique based on RBFs [56].

Coupling Schemes preCICE allows to decide at runtime whether a coupling scheme should be serial or parallel and whether it should be explicit or implicit. Here, serial refers to an execution order of two participants one after the other, parallel refers to a simultaneous execution. Explicit schemes only compute one step for every participant in every timestep, whereas implicit schemes sub-iterate until convergence. Serial-explicit and parallel-explicit correspond, thus, to the conventional schemes described in [83]. All four combinations can be combined with subcycling, meaning one participant performing multiple small timestep during one large timestep of the other participant.

The convergence and stability of implicit schemes can be improved by an acceleration technique – in preCICE nomenclature a post-processing scheme. preCICE offers static or dynamic-Aitken underrelexation [128] and more sophisticated quasi-Newton schemes. Two quasi-Newton approaches are supported: *interface quasi-Newton inverse least-squares* (IQN-ILS) [60], which corresponds to an Anderson acceleration [1], or *interface quasi-Newton multi vector Jacobian* (IQN-MVJ) [129], which corresponds to a generalized Broyden scheme [78]. Section 3.2 and 3.3 give detailed information on the schemes and Section 3.7 elaborates a numerical convergence study. The implementation on distributed data is discussed in Section 4.4.

To couple more than two participants, coupling schemes can be combined arbitrarily (composition of coupling schemes, first discussed in [99]) or can be be coupled fully-parallel (multi-coupling [44]). Section 3.8 discusses both variants.

Communication Three basic variants for communication are supported: MPI, TCP/IP sockets, implemented with `Boost.Asio`[4], or communication via files. The MPI communication allows for a start-up in a common communicator (referred to as `mpi-single`), based on `MPI_Comm_split` and `MPI_Intercomm_create`, or in separated communicators (simply referred to as `mpi`), based on the three methods `MPI_Open_port`, `MPI_Comm_accept`, and `MPI_Comm_connect`. These basic variants are all implemented in a 1:N fashion and can be used directly for communication between a server and all solver threads, such as in [99], or for communication between the master thread and all other solver threads. Furthermore, they can be used to build up M:N communications between participants. These come in two variants: a gather-scatter scheme, which is mainly intended for test purposes, and a fully point-to-point scheme. An M:N communication based upon the communication via files is not supported. Details on the M:N implementation and performance evaluations are given in Section 4.2.

2.1.3 Configuration

Earlier in this chapter, I mention that preCICE is configured at runtime via an xml file. In the driving example, Figure 18, this happens in line 3. In this section, I have a brief look on how this configuration file `precice-config.xml` is composed. I simply reuse the driving example and couple the `FluidSolver` to a `StructureSolver`. Figure 19 shows the example configuration. A brief reminder: `binprecice` can auto-generate a complete xml reference.

The complete configuration is encapsulated in the element `<precice-configuration>`, followed by the element `<solver-interface>`, where the dimension of the scenario is defined. In lines 3–14, the data fields and meshes are defined. Please note that these fields correspond to the names used in Figure 15. The participant `FluidSolver` is defined in lines 16–28. Here, line 17 defines the communication between the master and all slave threads of this participant. The next two lines specify that `FluidSolver` holds two meshes. One mesh, `FluidMesh` is provided by this participant. This means that `FluidSolver` needs to define the vertices of `FluidMesh` via `setMeshVertex` or `setMeshVertices`. The second mesh, `StructureMesh` is not defined by `FluidSolver`, but received from `StructureSolver`. Holding two meshes, the participant can now define a mapping between both. Here, radial basis function mappings with multiquadrics as basis functions, solved by PETSc, are used for both directions, compare lines 22–27. After the second participant, `StructureSolver`, is defined, line 37 specifies the communication between both participants. A point-to-point communication based on TCP/IP sockets is used. Finally, lines 39–58 define the coupling scheme between both participants. A parallel-implicit coupling with an IQN-ILS post-processing is used. Lines 41–42 show the timestep size and the maximum time. Lines 43–44 lists which data values are exchanged. Line 46–47 define the convergence measures. Lines 49–57 specify the details of the post-processing.

2.2 Developer Perspective

For the sake of completeness and also as the starting point for the implementation details in the next chapters, I also give a brief introduction to the developer perspective of preCICE. Of course, a more detailed description can be found in [99]. preCICE is composed of hierarchically structured components with a loose layering, cf. Figure 20. This means that each component can only access functionality from packages that lay on a lower level in the hierarchy. Table 1 lists all components together with a brief description of their purpose. This thesis benefits highly from the clean and clear software architecture of preCICE. New implementations can be added easily. I altered the component structure compared to [99] only in a minimal way – only the `m2n` package is added.

Figure 21 lists the structure of a single component. The interface of each component is accessible from other packages and is separated from its implementation in the subfolder `impl`. Each interface class has a corresponding unit test class in the subfolder `test`. The component `precice` holds integration

[4] `http://www.boost.org`

```xml
<precice-configuration>
  <solver-interface dimensions="3">
    <data:vector name="Displacements"/>
    <data:vector name="Forces"/>

    <mesh name="FluidMesh">
      <use-data name="Displacements"/>
      <use-data name="Forces"/>
    </mesh>

    <mesh name="StructureMesh">
      <use-data name="Displacements"/>
      <use-data name="Forces"/>
    </mesh>

    <participant name="FluidSolver">
      <master:mpi-single/>
      <use-mesh name="FluidMesh" provide="yes"/>
      <use-mesh name="StructureMesh" from="StructureSolver"/>
      <write-data name="Forces" mesh="FluidMesh"/>
      <read-data  name="Displacements" mesh="FluidMesh"/>
      <mapping:petrbf-multiquadrics shape-parameter="0.1" solver-rtol="1e-5"
        direction="write" from="FluidMesh" to="StructureMesh"
        constraint="conservative" timing="initial"/>
      <mapping:petrbf-multiquadrics shape-parameter="0.1" solver-rtol="1e-5"
        direction="read" from="StructureMesh" to="FluidMesh"
        constraint="consistent" timing="initial"/>
    </participant>

    <participant name="StructureSolver">
      <master:mpi-single/>
      <use-mesh name="StructureMesh" provide="yes"/>
      <write-data name="Displacements" mesh="StructureMesh"/>
      <read-data  name="Forces" mesh="StructureMesh"/>
    </participant>

    <m2n:sockets from="FluidSolver" to="StructureSolver" network="ib0" exchange-directory="../"/>

    <coupling-scheme:parallel-implicit>
      <participants first="FluidSolver" second="StructureSolver"/>
      <max-time value="1.0"/>
      <timestep-length value="1e-3" />
      <exchange data="Displacements" mesh="StructureMesh" from="StructureSolver" to="FluidSolver"/>
      <exchange data="Forces" mesh="StructureMesh" from="FluidSolver" to="StructureSolver"/>
      <max-iterations value="20"/>
      <relative-convergence-measure data="Displacements" mesh="StructureMesh" limit="1e-3"/>
      <relative-convergence-measure data="Forces" mesh="StructureMesh" limit="1e-3"/>
      <extrapolation-order value="2"/>
      <post-processing:IQN-ILS>
        <data name="Displacements" mesh="StructureMesh"/>
        <data name="Forces" mesh="StructureMesh"/>
        <initial-relaxation value="0.1"/>
        <max-used-iterations value="50"/>
        <timesteps-reused value="5"/>
        <filter type="QR1" limit="1e-6" />
        <preconditioner type="residual-sum" />
      </post-processing:IQN-ILS>
    </coupling-scheme:parallel-implicit>

  </solver-interface>
</precice-configuration>
```

Figure 19: An example configuration for a fluid-structure interaction coupling between the participant `FluidSolver` and the participant `StructureSolver`. This example matches the driving API example from Figure 18.

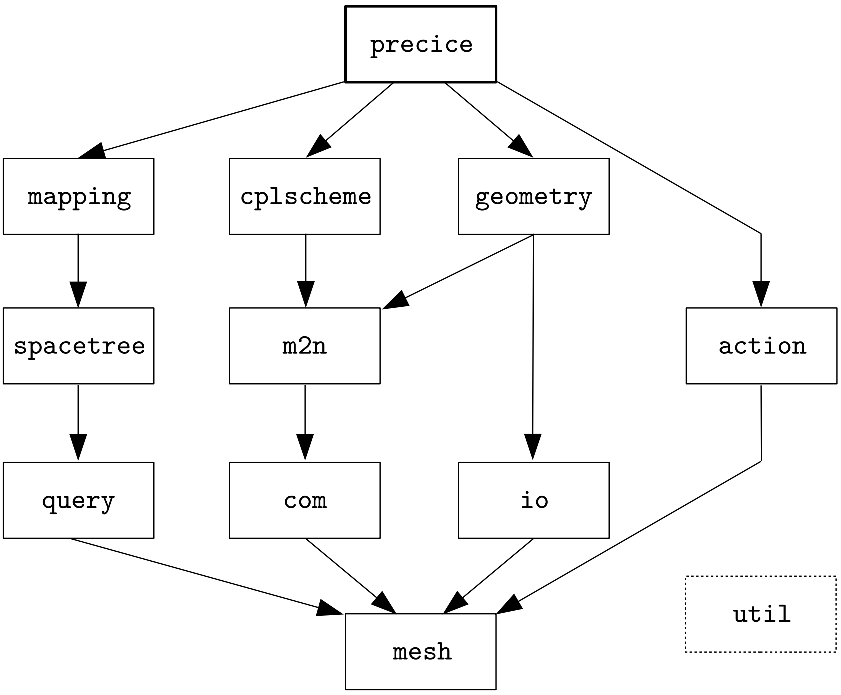


Figure 20: Loosely layered preCICE components with dependencies marked by arrows. A component might also access other components further down the dependency graph. Every component can access the component `util`. The package `m2n` is newly introduced in this thesis. A similar figure is also used in [99].

tests. Furthermore, the configuration is decentralized and implemented in the subfolder `config`. Various packages are extended in this thesis: the `geometry` package for the mesh repartitioning in Section 4.1.2, the `cplscheme` package for the parallel coupling schemes and the new post-processing methods in Section 3.5, the `mapping` package for PETSc-based RBF mappings. Section 4.2.2 details the `m2n` package and necessary changes in the `com` package.

2.3 Review of Alternative Coupling Software

After the brief introduction to preCICE from a user, but also from developer perspective in the last two sections, I now review other software for multi-physics coupling and draw some conclusions on where preCICE currently stands. Obviously, there is a vast amount of available multi-physics software. Thus, the first part of the review is to decide to what exactly I want to compare preCICE. As, surely, I want to compare preCICE to similar software, this becomes a question of what preCICE actually is.

2.3.1 Scope of the Review

Since preCICE is a coupling library for partitioned multi-physics, I exclude monolithic FSI software from the review and also general multi-physics packages. I solely want to focus on stand-alone coupling tools. I do not want to restrict the review solely to FSI, but also include other surface-coupled problems as preCICE can also be used for them, compare various examples in Chapter 5. For the sake of compactness, the review does not include multi-scale coupling tools. Multi-scale simulations share similar problems as multi-physics simulations, but have not exactly the same focus. [107] gives a thorough survey on such software. Furthermore, I include only library approaches to omit tools that do not have more or less the same goals as preCICE, such as the Uintah framework [143] or the SIERRA framework [191]. The grid-glue component of the DUNE framework [10] offers methods for interpolation and communication, but is not meant as a general tool for code coupling. Also, I exclude only partial solutions, such as the Component Template Library (CTL), which is used for FSI in [139], for example, since it solely offers communication means. The thesis of Bernhard Gatzhammer [99] gives a brief review on the latter

```
component /
   ClassA
   ClassB
   ...
   Constants

   impl /
      ImplementationClass1
      ImplementationClass2
      ...
   tests /
      TestCaseClassA
      TestCaseClassB
      TestCaseImplementationClass1
      TestCaseImplementationClass2
      TestCaseConfigurationClassA
      TestCaseConfigurationClassB
      ...
   config /
      ConfigurationClassA
      ConfigurationClassB
```

Figure 21: Substructure of a general component. The component interface lies directly at the root of the component. The implementation is separated in a subfolder `impl`. `tests` contains unit tests of all classes whereas `config` contains the decentralized configuration classes. A similar figure is also used in [99].

`action`	Auxiliary methods to modify coupling data or meshes at specific moments during a simulation. Built-in variants are offered as well the possibility to define new actions at run-time through Python callbacks.
`cplscheme`	Steers the time-dependent coupling between different participants. Offers methods for fix-point acceleration.
`com`	Provides basic 1:N communication means via TCP/IP, MPI, or files.
`geometry`	Methods to create meshes from built-in geometries, communicated geometries or loaded geometries.
`io`	Functionality to input and output mesh data structures or coupling states.
`m2n`	Constructs m:n communications between participants.
`mapping`	Provides methods to interpolate data between non-matching meshes.
`mesh`	Defines a mesh data structure consisting of vertices, edges, and triangles and associated data containers.
`precice`	Defines the application interface of preCICE.
`query`	Provides geometrical query operations for the mesh data structures, e.g. closest distances or volume queries for bounding boxes.
`spacetree`	Constructs spatial data structures to reduce the number of query operations.
`util`	Provides utility functionality, e.g. master-slave communication, debugging tracer, time measurement, etc.

Table 1: Brief description of all preCICE components.

software. Also, ASCoDT [3] only offers communication means. Other tools are no longer under active development, such as FSI*ce or CoMA, or were never meant to be community codes, such as FLECS or Tango. I skip a description of these, especially since they are already covered in [99].

For the reviewed tools, I rely solely on information given and did no exhaustive testing as this would be beyond the scope of this thesis. Information is gathered from publications, web pages, user manuals or brief looks into the code. Other software packages have different design goals than preCICE such that a fair evaluation is not always possible. Like mentioned above, [99] already presents a certain review on coupling software. Throughout the last couple of years, however, the landscape has changed significantly, such that an update is necessary. In particular, I want to focus this review on the main multi-physics goals derived in Chapter 1: flexibility and scalability. Parts of this review are also part of the master thesis of Alexander Shukaev [181]. Last, this list of tools is certainly not exhaustive, but relies on information that has been published the last couple of years, many conference discussions, and a thorough web search.

Aspects of the Review I evaluate the reviewed tools in terms of four aspects. The first aspect is the level of the API. I distinguish between a low level API, which operates on a similar level as MPI, an intermediate level, which is more flexible than a low level API, but which still uses explicit sending and receiving operations, and a high level API, which allows to configure arbitrary coupling schemes – serial or parallel, explicit or implicit, matching or non-matching timestep sizes – at run time. Of course, the transition between these levels is not always sharp. Next, I evaluate the HPC compatibility. A coupling tool is HPC compatible if it has no central instance as a bottleneck, compute-wise or communication-wise, but a clear parallel communication layout. Furthermore, I have a look at the legal situation, meaning whether the software is open-source, in-house, or commercial. Finally, not every coupling tool offers coupling schemes. I distinguish those which do from those which do not.

2.3.2 List of Tools

The tools are listed in alphabetic order.

ADVENTURE The ADVanced ENgineering analysis Tool for Ultra large REal world (ADVENTURE) is developed by the University of Tokyo and colaboration partners[5]. It is an environment of different solvers and different physics and claims high parallel efficiency. In this environment, the coupling library ADVENTURE_Coupler was recently added [121]. The tool is specifically designed to coupled the ADVENTURE fluid and structure solvers. The purpose is, thus, not generic. Many modules of the ADVENTURE project are open source, the ADVENTURE_Coupler, to my best knowledge, however not. The coupling library is server-based. The server is, however, parallelized. It runs on $M + N$ threads if N and M are the number of threads of the coupled solvers at the interface and establishes a parallel communication between the solvers and the coupler. [121] shows scalability results up to 64 threads. The API level is intermediate. ADVENTURE_Coupler offers coupling schemes such as a Broyden scheme and performs interpolation based on the element shape functions of both solvers.

Data Transfer Kit (DTK) and PIKE The DTK [184] is developed at the Oak Ridge National Laboratory as part of a broader coupling toolkit, which is generated by *The Consortium for Advanced Simulation of Light Water Reactors*. It uses a library approach and is open-source[6]. It offers methods for parallel communication. In particular, the initialization of the communication and the mesh partitioning are optimized such that it also allows for volume coupled problems. Interpolation can be based on simple projection schemes, but also on shape function interpolation or spline interpolation. The API level is low. DTK itself does not offer methods for equation coupling, but it is often combined with the PIKE package from the Sandia National Laboratories, which offers steering functionality and, for example, equation coupling based on Anderson acceleration. PIKE is part of Trilinos[7] and there are plans to include DTK as well.

[5] http://adventure.sys.t.u-tokyo.ac.jp/

[6] https://github.com/ORNL-CEES/DataTransferKit

[7] http://trilinos.org/

EMPIRE The coupling tool EMPIRE [219, 182] is the successor of CoMA and uses a library approach. A centralized serial server is used to compute interpolation schemes and coupling schemes. EMPIRE suffers therefore from the same bottlenecks as the server-based version of preCICE, compare Section 1.3. Interpolation can be a simple projection scheme, but also based on NURBS or Mortar approaches. EMPIRE offers a methodology for multi-coupling, which can also deal with Neumann-Neumann or Dirichlet-Dirichlet couplings (compared to the classical Dirichlet-Neumann coupling, detailed in Section 3.1) and ODE signals. It is open-source[8] and has an intermediate API level.

MpCCI The Fraunhofer Institute for Algorithms and Scientific Computing (SCAI) develops the coupling tool MpCCI[9] [119], which has become the commercial standard for FSI coupling software. It was one of the first successful coupling libraries, available already since 2002. It can be seen as a mixture between a library and a framework approach, since a library can be used for steering, but the data exchange works in a more framework-like way. MpCCI works mainly with ready-to-use adapters for commercial solvers and standard open-source solvers, but it also offers a C++ API, which can be included in other codes. In contrast to nearly all other academic software, it offers a graphical user interface to control and overview the simulation progress. Interpolation schemes are based on shape functions or projection. While MpCCI offers, since recent years, implicit coupling schemes, it lacks sophisticated acceleration schemes. Also, to my best knowledge, multi-coupling is not possible in a straight-forward way. On the other hand, coupling to 1D or ODE problems is possible. The tool is based on a serial server, which leads to the usual scalability limitations. This is, however, also not the primary focus of MpCCI. Communication works solely with TCP/IP, MPI is not supported. The latter might, however, also be problematic, since MpCCI is a binary distributed software, which could lead to MPI consistency issues when linked with other solvers.

OASIS3 and Model Coupling Toolkit (MCT) CERFACS in Toulouse and the *Centre National de la Recherche Scientifique* in Paris co-develop the coupling tool OASIS3, which is mainly designed for massively parallel climate modeling [205]. The tool is build upon the Model Coupling Toolkit (MCT) from the Argonne National Laboratory and is open-source[10]. It uses a library approach and offers interpolation and communication both in parallel. Furthermore, OASIS3 offers a methodology to interpolate in time. The API is at a low level. In particular, no coupling schemes are supported.

OpenPALM and CWIPI CERFACS and ONERA also co-develop the open-source coupling library OpenPALM[11], which uses the library CWIPI[12] as a backbone. OpenPALM features parallel communication based on MPI and also a parallel setup phase. [73] shows scalability results up to 12000 threads. Interpolation can be done via projection schemes. In particular, there is a possibility to define own higher order projection schemes via a callback functionality. A GUI named PrePALM allows to establish the coupling and monitor the simulation at run-time. OpenPALM features a high-level API and offers, to my best knowledge, no coupling schemes.

PLE *Électricité de France* (EDF) develops the open-source fluid solver Code_Saturne[13], which also features the coupling library Parallel Location and Exchange (PLE)[14]. PLE claims to perform parallel mapping and communication, a thorough evaluation is, however, not possible due to limited information provided.

Conclusions Table 2 lists the evaluation of all reviewed tools compared to preCICE. The main design goal of preCICE is to provide a minimal time-to-solution for legacy codes, where time-to-solution refers to the complete time spent in the development and setup of a multi-physics simulation. Therefore, preCICE features black-box coupling capabilities and a high-level API. The latter not only simplifies

[8] `http://www.empire-multiphysics.com/`
[9] `http://www.mpcci.de/mpcci-software.html`
[10] `https://verc.enes.org/oasis`
[11] `http://www.cerfacs.fr/globc/PALM_WEB/index.html`
[12] `http://sites.onera.fr/cwipi/`
[13] `http://code-saturne.org/cms/`
[14] `http://code-saturne.org/doxygen/src/ple/index.html`

	API Level	HPC	Legal	Coupling Schemes
ADVENTURE	intermediate	yes	in-house	yes
DTK (+PIKE)	low (DTK) / high (PIKE)	yes	open source	yes (PIKE)
EMPIRE	intermediate	no	open source	yes
MpCCI	intermediate	no	commercial	yes
OASIS3	low	yes	open source	no
OpenPALM	high	yes	open source	no
preCICE	high	yes	open source	yes

Table 2: Summary of other coupling libraries compared to preCICE in several aspects. Section 2.3.1 gives an explanation of the evaluation of the aspects.

the first integration, but also the testing phase until an application runs in a stable way due to the higher flexibility at run time. Thus, it comes at no surprise that preCICE offers a higher API level than most other tools. Meanwhile, most coupling libraries offer parallel communication and interpolation to support massively parallel solvers. A unique selling point of preCICE might be the sophisticated coupling schemes which are also fully parallel. Some limitations of preCICE compared to other software are the missing support of volume coupling and ODE coupling. Furthermore, industrial relevance has not yet been proven. Finally, coupling to particle codes is not included.

2.4 Used Single-Physics Solvers

This section gives brief descriptions of all single-physics solvers that are used in this thesis as test or show cases. Furthermore, all other solvers that are currently, to my knowledge, coupled to preCICE are listed. I give no exhaustive description of the physical modeling or the numerical schemes of each solver, but refer the reader to the corresponding references. Table 3 gives an overview of all coupled solvers.

Alya System The Alya System is a finite element multi-physics code developed at the Barcelona Supercomputing Center[15]. The different physics are organized in modules such as Nastin or Nastal for incompressible or compressible flow, respectively, Solidz for non-linear solid mechanics or Alefor for mesh deformation. Furthermore, there are modules for species transport equations, excitable media, heat transport, n-body collision, electro-magnetism, quantum mechanics, and Lagrangian particle transport. The preCICE adapter [203] is written in a general form, such that, in principle, it is usable with any module. The focus however lies on Nastin, internally coupled to Alefor, and Solidz. Nastin and Solidz are part of the benchmark suite of the Partnership for Advanced Computing in Europe (PRACE)[16], and thereafter open-source to some extent, but in principle Alya is an in-house code. Alya uses unstructured grids and a domain decomposition based on METIS[17]. The parallel performance of Alya is analyzed in [115] and good weak scalability up to 100,000 cores is shown in [213]. Nastin uses a stabilized finite element formulation based on the varitational multi-scale method [116] and the sub-grid scale tracking technique described in [114]. To resolve the equations, Orthomin(1), a fractional step technique, is used for the Schur complement of the pressure [113]. [201] gives an overview on turbulence models that can be coupled to Nastin. Nastin can be used in an ALE formulation. Then, Alefor describes the mesh movement, governed by a simple Laplace equation with pseudo-physical properties to preserve the quality of the mesh in boundary layers. Solidz uses constitutive equations for small and large deformation. Various elasticity models are available. Discretization is solely based on lower order finite elements, no shell elements are available. For both modules, Nastin and Solidz, various explicit and implicit time-integration schemes are implemented.

Ateles The group for Simulation Techniques and Scientific Computing of the University Siegen (STS) develops the in-house discontinuous Galerkin solver Ateles [239, 238]. The solver uses a pure explicit time integration scheme up to order four and is specifically designed for high order spatial elements.

[15] http://www.bsc.es/es/computer-applications/alya-system

[16] http://www.prace-ri.eu/ueabs

[17] http://glaros.dtc.umn.edu/gkhome/views/metis

Solver	Physics	View-Point	Discr.	Legal	Group
Ateles Acoustics	A	Eulerian	DG	in-house	U Siegen
Ateles Euler	CF	Eulerian	DG	in-house	U Siegen
Ateles Navier-Stokes	CF	Eulerian	DG	in-house	U Siegen
Alya Nastin	IF	ALE	FE	in-house	BSC
Alya Solidz	S	Lagrangian	FE	in-house	BSC
A*STAR Flow	CF	ALE	FV	in-house	A*STAR
Calculix	S	Lagrangian	FE	open-source	A*STAR
Carat	S	Lagrangian	FE	in-house	TUM STATIK
COMSOL	S	Lagrangian	FE	commercial	TUM SCCS
EFD	IF	Eulerian	FD	in-house	TUM SCCS
FASTEST	IF+A	ALE	FV	in-house	TU Darmstadt
FEAP	S	Lagrangian	FE	in-house	TU Darmstadt
FEM-shell	S	Lagrangian	FE	open-source	U Stuttgart SGS
Fluent	IF	ALE	FV	commercial	TUM SCCS
OpenFOAM	CF	ALE	FV	open-source	U Delft
OpenFOAM	IF	ALE	FV	open-source	U Delft
OpenFOAM	S	Lagrangian	FV	open-source	U Delft
Peano1	IF	Eulerian	FE	in-house	TUM SCCS
Structure0815	RB	Lagrangian	-	in-house	TUM SCCS
SU2	CF	ALE	FV	open-source	TUM SCCS

Table 3: List of single-physics solvers that are currently coupled to preCICE. Physics: A – acoustics (linearized Euler equations), CF – compressible flow (Euler or Navier-Stokes equations), IF – incompressible Navier-Stokes equation, RB – rigid body movement, S – structural mechanics. Discretization (in space): DG – discontinuous Galerkin, FD – finite differences, FE – finite elements, FV – finite volumes. Group: either the group which develops the solver or, for community or commercial codes, the group which develops the preCICE adapter.

Ateles is part of the APES suite [123], which holds amongst various solvers also tools for pre- and postprocessing, all based on the structured adaptive mesh library `TreElM`[18]. The APES framework is designed to run efficiently on massively parallel systems and uses a space-filling curve to minimize the required storage and to maximize data locality. Ateles shows a very good strong scalability down to a single element per core [123]. An embedded high order representation of material properties allows to resolve complex geometries. The preCICE adapter is developed by Verena Krupp and documented in [41].

OpenFOAM OpenFOAM is an open-source finite volume solver[19]. The preCICE adapter, which is also freely available[20], is written by David Blom from the Technical University Delft and build upon the foam-extended-3.1 fork[21]. OpenFOAM is originally designed as a solver for the incompressible Navier-Stokes equations, but also implementations for compressible flow and non-linear elasticity [48] are available. OpenFOAM uses unstructured grids. The flow solver is based on a second order implicit time integration combined with a fully implicit pressure-velocity solver. The equations are formulated in the ALE setting, while the mesh deformation is solved by an RBF interpolation. The structure solver is based on the Saint-Venant-Kirchhoff hyperelastic constitutive relation to allow for large displacements and is also second order in time.

Carat The STATIK group[22] from the Technical University of Munich develops the in-house structural mechanics code Carat++ [24, 92]. Just as the structure codes from Alya and OpenFOAM, Carat++ is also based on the Saint-Venant-Kirchhoff hyperelastic constitutive relation. The code is, however,

[18] `https://bitbucket.org/apesteam/treelm`

[19] `http://www.openfoam.org`

[20] `https://github.com/davidsblom/FOAM-FSI`

[21] `http://www.extend-project.de`

[22] `https://www.st.bgu.tum.de/en/lehre0/research/carat/`

developed with an emphasis on the prediction of shell or membrane solver and offers various finite element shape functions for this purpose. Special focus is put on form finding and non-linear dynamic problems. Different time-integration schemes are available such as, for example, the implicit generalized-α method.

SU2 The Aerospace Design Laboratory of Stanford University initiated the open-source[23] CFD code SU2 – Stanford University Unstructured[24]. The code is developed from an aerodynamical point of view, including classical CFD analysis, but also design-driven tasks such as shape optimization. SU2 uses a dual-mesh finite volume method [158]. Currently, only the compressible flow module is coupled to preCICE as it is the only module that supports an ALE moving mesh technique. The preCICE adapter of SU2 is described in [172], along with a detailed description of SU2 as well as the adapter.

Other Solvers Various other solvers are coupled to preCICE. I only list these solvers here, without a deep description, due to the fact that they are not used for any tests in this thesis. The TU Darmstadt develops the incompressible flow solver FASTEST, which uses a splitting scheme to also resolve acoustic phenomena [126]. The same group coupled FEAP[25], a commercial finite element structure solver. In the thesis of Bernhard Gatzhammer [99] as well as in [142], the commercial solvers Ansys Fluent and COMSOL are used. [99] uses furthermore the rigid body solver Structure0815, which is included in the preCICE source tree, and Peano1 [45, 152], an incompressible flow solver which is developed at the chair for Scientific Computing in Computer Science (SCCS), Technical University of Munich, and is open-source. The Institute of High-Performance Computing (IHPC) at the A*STAR institute in Singapore adapted the open-source solver Calculix for plastic deformations and their own in-house flow solver [153]. Finally, Viacheslav Mikerov and Stephan Herb developed in their master theses the fixed-grid, finite difference, incompressible flow solver EFD[26] [147] and the finite element shell solver FEM-shell[27] [112], respectively.

Summary of Chapter 2

- preCICE enables the coupling of black-box single-physics solvers at runtime.
- To this end, methods for interpolation between non-matching coupling meshes, communication means between separate executables, and fixed-point acceleration schemes are provided.
- To prepare a solver for coupling via preCICE, an adapter needs to be written. I give a general example, which has less than 40 additional lines of code.
- preCICE uses a layered package structure, which facilitates the introduction of new features.
- I compare preCICE to other coupling tools. A unique selling point of preCICE is the high-level API besides sophisticated quasi-Newton coupling schemes.
- preCICE is already used by various in-house and open-source solvers.

[23] https://github.com/su2code/SU2

[24] http://su2.stanford.edu

[25] http://www.ce.berkeley.edu/projects/feap/

[26] https://github.com/precice/efd

[27] https://github.com/precice/fem-shell

3 Inter-Solver Parallelism: Parallel Coupling Schemes

In the introduction of this thesis, I mention several ingredients for a scalable partitioned approach (Section 1.3). One of these ingredients is a coupling scheme that allows for a simultaneous execution of the fluid and the structure solver. Due to the load imbalance of both solvers, this is the only clean strategy to avoid idling processors. In this chapter, I describe such parallel coupling schemes, along with a general introduction to coupling schemes. The focus of this chapter is a numerical one. This means that I essentially rely on iteration numbers to judge the efficiency of algorithms. For practical applications, however, many different aspects determine the final efficiency of an algorithm. Load-balancing between the fluid and structure solver, for example, plays a crucial role. Such issues, including run-time comparisons, are exemplarily studied in Section 5.2.

Besides the description of parallel coupling schemes, a further contribution of this chapter is the systematic comparison of interface quasi-Newton schemes, which have been a topic of research in the partitioned fluid-structure interaction (FSI) community over the past decade, with the acceleration techniques that go back to the original paper of Donald G. Anderson from 1965 [1]. These acceleration methods regained new attention over the last decade as well as the methods are promising if applied to multi-physics problems or, more general, if applied to legacy codes. Until very recently, probably 2014, the two communities, the partitioned FSI community and the Anderson acceleration (AA) community were, to my best knowledge, not aware of the similarities of their methods.

Another aspect of this chapter is the reduction of parameters of the coupling schemes. As mentioned in the introduction, the success of multi-physics simulations highly depends on the complexity and thereby the usability of each of its components. This chapter describes a coupling scheme variant, based on a generalized Broyden scheme, that allows for an implicit reuse of information from past timesteps and, thus, renders an explicit tuning parameter dispensable. Furthermore, I describe efforts for an automatic numerical scaling of the parallel coupling system in this chapter.

Finally, the generalization of parallel coupling schemes allows for a stable coupling of multi-physics scenarios with more than two components. To my best knowledge, this is the first method that allows for such a partitioned strong multi-problem coupling, although similar work was documented independently for FSI in [182] and for pellet-cladding interaction in [197].

Just as in the last chapters, I want to mention my collaborators. Besides Miriam Mehl, I worked with David Blom and Bernhard Gatzhammer on the testing of the parallel coupling schemes. We published first results in [202] and more detailed results in [142]. In the master thesis of Klaudius Scheufele [176], we worked on the generalized Broyden coupling schemes, which resulted in a joint publication [129]. Also, the structure of this chapter follows [176] to some extent. A review on our work on coupling schemes was published in [27]. The studies concerning the scaling of the parallel system were developed during my stay at the Lawrence Livermore National Lab, a joint work of John Loffeld, Carol Woodward and myself. The filtering techniques were a joint effort of Alfred Bogaers, Robby Haelterman, Miriam Mehl, Klaudius Scheufele, and myself, and were published in [109]. The generalization of the parallel coupling schemes to multi-coupling scenarios were published in [204, 44].

This chapter is structured as follows: Section 3.1 introduces the general mathematical setup, including the fixed-point equation systems that describe the physical coupling conditions at the fluid-structure interface. As the second building block, Section 3.2 gives a detailed introduction to fixed-point equation solvers, including a broad literature review. Both building blocks are then combined in Section 3.3 to formulate concrete coupling schemes for FSI, which are first tested in Section 3.4 by means of a simple 1D testcase to reduce the zoo of methods to the most efficient ones. Section 3.5 then describes the implementation of those methods in the coupling library preCICE. Section 3.6 gathers the advanced topics filtering and preconditioning of the coupling schemes. Advanced numerical results are shown in Section 3.7. Finally, Section 3.8 explains how to generalize parallel coupling schemes to multi-coupling schemes.

3.1 Ingredients of the Partitioned Coupling Approach

This section introduces basic ingredients for a black-box fluid-structure coupling. As we have no influence to or details from the internals of a black-box solver, we abstract the discretized solvers to simple operators on the coupling interface:

$$F : \mathbb{R}^n \to \mathbb{R}^n, d \mapsto f \;\text{ and }\; S : \mathbb{R}^n \to \mathbb{R}^n, f \mapsto d \, .$$

F refers to the computation of one fluid timestep. This means displacements d, relative to the last timestep, are read from interface, transformed to velocity and set as Dirichlet boundary conditions. Then, one fluid timestep is solved and forces or tractions f are computed. The structure solver S, on the other hand, reads f, uses it as Neumann boundary conditions and computes one timestep. This coupling approach is, therefore, referred to as Dirichlet-Neumann coupling. Both coupling variables d and f live in the same discretized space, meaning on the same vertices (n is the number of vertices times the dimension of the scenario). In the case of non-matching grids at the coupling interface, necessary mapping operators are part of F or S and, thus, hidden in this notation, see also Section 4.3. The same applies if, for example, the fluid solver discretizes velocity and force values at different grid points. Furthermore, a strategy to deal with moving boundaries, an arbitrary Lagrangian-Eulerian approach or an immersed boundary method are also part of F.

Fixed-Point Systems As mentioned in the introduction, the partitioned coupling approach suffers inherently from the added-mass effect. To control instabilities, a solution of equal quality than a monolithic solution has to be recovered in every timestep. This encompasses the kinematic interface condition, the balance of displacements and velocities, and the dynamic interface condition, the balance of forces. These conditions lead to fixed-point equations, which need to be resolved by a sub-iteration process.

The classical fixed-point system is based on a Gauß-Seidel execution of both solvers:

$$(S \circ F)(d) = d \, . \tag{GS}$$

A multiplication from left by F yields the force balance. This fixed-point equation strictly infers a sequential execution of both solvers and therefore suffers from the aforementioned parallel efficiency drawback. Sometimes, this system is also referred to as serial or sequential system.

The straightforward way to allow for a simultaneous execution of both solvers is to reformulate the fixed-point equation as a Jacobi system:

$$\begin{pmatrix} 0 & S \\ F & 0 \end{pmatrix} \begin{pmatrix} d \\ f \end{pmatrix} = \begin{pmatrix} d \\ f \end{pmatrix} . \tag{J}$$

This system is also referred to as vectorial system. An alternative parallel formulation is based on the inversion of the structure operator, which is a standard operation in contrast to the inversion of the fluid operator. Starting from identical displacement values, the force balance reads

$$F(d) = S^{-1}(d) \, ,$$

which can be transformed to a fixed-point equation[28]

$$F(d) - S^{-1}(d) + d = d \, . \tag{SP}$$

This system has been used in [66] under the name of Steklov-Poincaré and is also referred to as parallel system. Finally, for sake of completeness, I also introduce the block-iterative system or simple B-system, which is used in [216]:

$$\begin{pmatrix} S \circ F & 0 \\ 0 & F \circ S \end{pmatrix} \begin{pmatrix} d \\ f \end{pmatrix} = \begin{pmatrix} d \\ f \end{pmatrix} . \tag{B}$$

This system is to some extent connected to (J), as it is equal to the squared operator of (J). A block-iterative solution of this system leads, however, to a sequential execution of both solvers.

To unify the further studies, we abstract the four introduced systems into one general notation:

$$H : \mathbb{R}^n \to \mathbb{R}^n, \; H(x) = x \, . \tag{H}$$

[28] The fixed-point equation is not unique as the force balance could, of course, be multiplied by any $\gamma \neq 0$. To this end, differences in scale of the force and the displacement values could be numerically adjusted.

In the remainder of this chapter, an upper index k denotes the iterative procedure to resolve any fixed-point equation per timestep: $x^0, x^1, \dots, x^k$. A tilde denotes the Picard iterate $\tilde{x}^k = H(x^k)$. All introduced methods of the next section follow the same two-step outline. First, a Picard step is computed, and then, second, accelerated to find the next iterate:

$$x^k \overset{\text{Picard}}{\rightsquigarrow} \tilde{x}^k \overset{\text{Accel.}}{\rightsquigarrow} x^{k+1} \,.$$

Figures 22 to 25 review and compare the four introduced fixed-point equations under this notation and show the underlying data flow and execution order of the solvers.

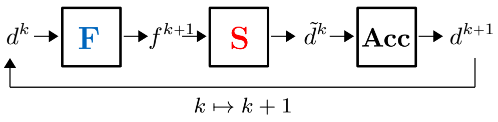


Figure 22: Flow chart for (GS)

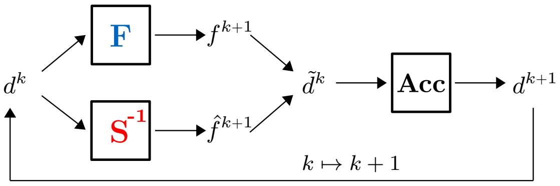


Figure 23: Flow chart for (SP)

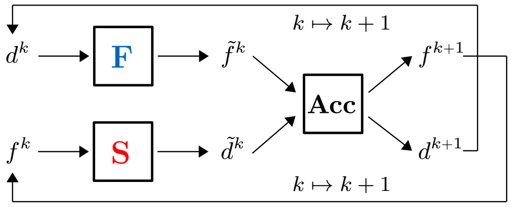


Figure 24: Flow chart for (J)

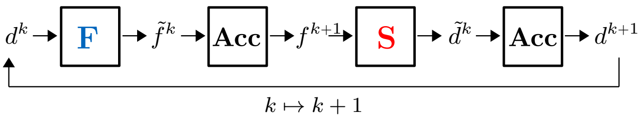


Figure 25: Flow chart for (B)

In this chapter, I assume that all costs involved into the acceleration are negligible compared to the Picard step. This is reasonable, as the Picard step typically consists of solving non-linear problems on the complete domain, whereas the acceleration shrinks to cheap problems at the coupling interface, a by one dimension smaller space. Therefore, I compare different acceleration techniques in the following by simple iteration counts per timestep.

Convergence Criteria The formulation of convergence criteria is a non-trivial task, since it highly influences the performance judgment of each method. To only look at convergence criteria built on the abstract notation (H) yields no fair comparison since this leads, for example, to a pure displacement check for (GS), but to a displacement and force check for (J). In particular, the fact that the convergence of (GS) is only checked by a displacement measure is a pure numerical conclusion, not a physical one. I conclude from this observation that a fair convergence criterion must be based on a solely physical argument. Therefore, I always check for displacements and forces. This is also the closest possible criterion to a monolithic formulation, following the guideline *coupled until proven decoupled* [122]. As, furthermore, displacements and forces might live on rather different scales, a purely relative measure is used. For sake of a clean understanding, I detail this measure for every fixed-point equation:

(GS)	$\|f^{k+1} - f^k\|_2 < \epsilon_{rel} \cdot \|f^{k+1}\|_2$	&	$\|\tilde{d}^k - d^k\|_2 < \epsilon_{rel} \cdot \|\tilde{d}^k\|_2$
(J)	$\|\tilde{f}^k - f^k\|_2 < \epsilon_{rel} \cdot \|\tilde{f}^k\|_2$	&	$\|\tilde{d}^k - d^k\|_2 < \epsilon_{rel} \cdot \|\tilde{d}^k\|_2$
(SP)	$\|f^{k+1} - f^k\|_2 < \epsilon_{rel} \cdot \|f^{k+1}\|_2$	&	$\|S(f^{k+1}) - d^k\|_2 < \epsilon_{rel} \cdot \|S(f^{k+1})\|_2$
(B)	$\|f^{k+1} - f^k\|_2 < \epsilon_{rel} \cdot \|f^{k+1}\|_2$	&	$\|\tilde{d}^k - d^k\|_2 < \epsilon_{rel} \cdot \|\tilde{d}^k\|_2$

The convergence criterion for (SP) includes the solution of a further structure step to enable a fair comparison. In production use, this step should be avoided. Please note that the choice on this *outer* convergence measure also influences the choice of the convergence measure of a possible *inner* iteration in F or S. In particular, an inner convergence measure must be sufficiently tight. For more information, the reader may consult the general concept in [77] or the specific FSI consideration in [20].

3.2 Fixed-Point Equation Solvers

The simplest way to solve (H) is a pure Picard iteration, i.e. no acceleration is applied. This is sometimes referred to as multiplicative Schwarz procedure for (GS), respectively additive Schwarz procedure for (J). Numerous studies show that this does not yield any stability improvement if no relaxation is applied, compare, e.g., [218, 128]. The relaxed Picard iteration reads

$$x^{k+1} = \omega^k \cdot \tilde{x}^k + (1 - \omega^k) \cdot x^k \ .$$

While a constant (under-)relaxation paramter $\omega^k \in]0,1]$ might be sufficient for certain applications, a dynamic Aitken choice [118] is widely used,

$$\omega^k = -\omega^{k-1} \frac{(R^{k-1})^T \left(R^k - R^{k-1}\right)}{\|R^k - R^{k-1}\|_2^2} \ ,$$

with $R^k := \tilde{x}^k - x^k$ denoting the residual. For most applications, such an Aitken underrelexation, is however outdated, since outperformed by the more sophisticated quasi-Newton coupling schemes. These methods are based on multi-secant methods, whose general concepts the following section introduces. Afterwards, Section 3.2.2 details the most relevant two examples: Anderson acceleration and the generalized Broyden method.

3.2.1 Introduction to Multi-Secant Methods

Incorporating previous information beyond just the last iterate allows for the construction of more sophisticated methods. Fang and Saad introduced the following characteristics for non-linear problems $R(x) = H(x) - x = 0$ for which multi-secant methods are the best choice [78]:

1. The dimension of R is large, $n \gg 0$.
2. R is continuously differentiable, but the analytic from of its Jacobian is not accessible (or simply too expensive to compute).
3. The cost of evaluating $R(x)$ is high.
4. The problem is noisy, i.e. the evaluation of $R(x)$ contains errors.

Except the first characteristic, the partitioned FSI problem falls into this class of problems. This first characteristic has mainly implications in the storage requirements, but not in the overall design of the algorithm, for which characteristics two and three are dominant. The fourth characteristic is a fuzzy one as too much noise renders multi-secant methods unusable. We revisit this issue later in this section. For the following considerations, I follow the multi-secant viewpoint of Fang and Saad [78], but stick to the familiar FSI notation, of e.g. [60].

An inversion of the non-linear problem allows to also include the newest Picard step,

$$\tilde{R}(\tilde{x}) := \tilde{x} - H^{-1}(\tilde{x}) \overset{!}{=} 0 \ .$$

Let us assume that we have already successfully computed $k \geq 2$ iterates, after the computation of the first step by an underrelaxtion. The Newton update from the Picard step $\tilde{x}_k$ to the next iterate x_{k+1} reads

$$\textbf{solve} \quad \left[I - J_{H^{-1}}(\tilde{x}^k)\right] \Delta\tilde{x}^k = J_{\tilde{R}}(\tilde{x}^k)\Delta\tilde{x}^k = -\tilde{R}(\tilde{x}^k) \ , \tag{1}$$

$$\textbf{set} \quad x^{k+1} = \tilde{x}^k + \Delta\tilde{x}^k \ . \tag{2}$$

As the Jacobian $J_{\tilde{R}}(\tilde{x}^k)$ is not available, we construct an approximation based on input-output information from past iterates of H, stored in the tall and skinny matrices

$$\begin{aligned} W_k = \ & (w^i)_{i=0}^{k-1} = \left[\Delta\tilde{x}^0, \Delta\tilde{x}^1, \ldots, \Delta\tilde{x}^{k-1}\right], \qquad \text{with } \ \Delta\tilde{x}^i = \tilde{x}^{i+1} - \tilde{x}^i \ , \\ V_k = \ & (v^i)_{i=0}^{k-1} = \left[\Delta R^0, \Delta R^1, \ldots, \Delta R^{k-1}\right], \qquad \text{with } \ \Delta R^i = R^{i+1} - R^i \ . \end{aligned}$$

We seek for the next update $\Delta\tilde{x}^k$ in the column space of W_k. If W_k was formulated in terms of x^k and not $\tilde{x}^k$, the search space would always stagnate at the original dimension 2 (start-value and underrelaxed first iterate). Therefore, it is a crucial ingredient to formulate the Newton update from the Picard step $\tilde{x}^k$ to the next iterate x^{k+1} and not directly from x^k to x^{k+1}.

FSI applications make use of fixed-point algorithms in a transient setting, i.e. we solve a fixed-point problem in every timestep. For multi-secant methods, the reuse of information from past timesteps is important for the efficiency of the applied algorithms. By adding further blocks to W_k and V_k, we can explicitly reuse past information:

$$\begin{aligned} W_k^{(R)} &= \left[W^{(N+1-R)}, W^{(N-R)}, \cdots, W_k^{(N+1)}\right] , \\ V_k^{(R)} &= \left[V^{(N+1-R)}, V^{(N-R)}, \cdots, V_k^{(N+1)}\right] . \end{aligned}$$

An upper index (N) refers to the timestep throughout this work and is typically suppressed for the current timestep $N+1$. $W^{(N)}$ respectively $V^{(N)}$ then refer to the complete matrices after convergence in timestep N. The above matrices reuse, thus, information from R previous timesteps. Please note that information from different timesteps is not consistent as the underlying fixed-point equation changes. It is, therefore, necessary that the multi-secant method can handle such noise to some moderate extent. In the following, I suppress the upper index (R) and refer to the dimension of W_k and V_k as $\in \mathbb{R}^{n\times k}$ for the sake of readability.

By a direct approximation of the inverse Jacobian, we can render the solution of the Newton update (1) needless. The multi-secant equation for the inverse Jacobian reads

$$J_{\tilde{R}}^{-1}(\tilde{x}^k)\; V_k \approx W_k \; . \tag{3}$$

To approximate the inverse Jacobian, we solve (3), enhanced by further conditions to establish a unique solution. The choice of further conditions leads to different methods, which the next section elaborates. Please note that I refer to the exact Jacobian and the approximate Jacobian with the same notation as one can always easily discriminate between both from context.

3.2.2 Anderson Acceleration and Generalized Broyden

This section first introduces the basic principles of Anderson acceleration and the generalized Broyden method before diving into a broader literature review discussing applications as well as theoretical properties of both methods.

Anderson Acceleration Let us seek for a minimal norm approximate Jacobian, which fulfills the multi-secant equation (3) and

$$\|J_{\tilde{R}}^{-1}(\tilde{x}^k)\|_F \rightarrow \min \; .$$

Adding the side condition (3) as a Lagrange multiplier reads

$$L(J_{\tilde{R}}^{-1}, \lambda) := \frac{1}{2}\|J_{\tilde{R}}^{-1}\|_F^2 + \lambda \cdot (J_{\tilde{R}}^{-1} V_k - W_k) \rightarrow \min \; .$$

L denotes the Lagrange functional and $\lambda \in \mathbb{R}^k$ the Lagrange multiplier. The solution $(\bar{J}^{-1}, \bar{\lambda})$ has to fulfill the optimality conditions

$$\begin{aligned} \nabla_{J_{\tilde{R}}^{-1}} L|_{(\bar{J}^{-1},\bar{\lambda})} &= \bar{J}^{-1} + \bar{\lambda} \cdot V_k^T \overset{!}{=} 0 \; , \\ \nabla_{\lambda} L|_{(\bar{J}^{-1},\bar{\lambda})} &= \bar{J}^{-1} V_k - W_k \overset{!}{=} 0 \; . \end{aligned}$$

Inserting the first equation into the second one gives

$$\bar{\lambda} = -W_k (V_k^T V_k)^{-1} \; ,$$

and hence

$$\bar{J}^{-1} = W_k (V_k^T V_k)^{-1} V_k^T \; .$$

A QR-decomposition of V_k allows to implement this method as a matrix-free version as detailed in Algorithm 1, left. This also shows the equivalence to the least-square problem

$$\alpha = \mathrm{argmin}_{\hat{\alpha}\in\mathbb{R}^k} \|V_k \hat{\alpha} + R^k\|_2 \rightsquigarrow \Delta\tilde{x}^k = W_k \alpha \,,$$

which is, for example, used in [60] to derive the same method.

Generalized Broyden One drawback of Anderson acceleration is the choice upon R, the number of reused timesteps, which is unclear and highly problem dependent. This is the starting point for the generalized Broyden method. By the condition

$$\|J_{\tilde{R}}^{-1}(\tilde{x}^k) - J_{\tilde{R}}^{-1,(N)}(\tilde{x}^k)\|_F \to \min \,,$$

we keep the recent Jacobian close to the one of the previous timestep $J_{\tilde{R}}^{-1,(n)}$. Hereby, we hope to capture information from past timesteps in an implicit fashion, rendering the parameter R dispensable. Analog to the Lagrange calculation for the Anderson acceleration above, we can derive an update formula for the Jacobian:

$$J_{\tilde{R}}^{-1} = J_{\tilde{R}}^{-1,(N)} + (W_k - J_{\tilde{R}}^{-1,(N)} V_k)(V_k^T V_k)^{-1} V_k^T \,.$$

In practice, again, a QR-decomposition is applied, which, however, does not allow for a matrix-free implementation in this case, as there is no remedy for storing the previous Jacobian. Still, a QR-decomposition is favored over the solution of the normal equation, since V^k tends to be rank-deficient (compare [217]). Section 3.6.2 shows more details on this topic. Algorithm 1, right, lists the details. In [176], variants based on a multi-secant equation that span several timesteps as well one that spans only a fixed amount of columns are studied, without efficiency benefits.

Algorithm 1 Quasi-Newton methods in pseudocode. Underrelaxation with $\omega \in [0;1]$ is only performed in the first timestep.

```
Anderson Acceleration
initial value $x^0$
$\tilde{x}^0 = H(x^0)$ and $R^0 = \tilde{x}^0 - x^0$
$x^1 = x^0 + \omega \cdot R^0$
for $k = 1 \dots$ do
    $\tilde{x}^k = H(x^k)$ and $R^k = \tilde{x}^k - x^k$
    $V_k = [\Delta R^0, \dots, \Delta R^{k-1}]$, $\Delta R^i = R^{i+1} - R^i$
    $W_k = [\Delta \tilde{x}^0, \dots, \Delta \tilde{x}^{k-1}]$, $\Delta \tilde{x}^i = \tilde{x}^{i+1} - \tilde{x}^i$
    decompose $V^k = QU$
    solve $U\alpha = -Q^T R^k$
    $\Delta\tilde{x} = W_k \alpha$
    $x^{k+1} = \tilde{x}^k + \Delta\tilde{x}^k$
end for
```

```
Generalized Broyden
initial value $x^0$, $J_{\tilde{R}}^{(N),-1}$ from prev. timestep
$\tilde{x}^0 = H(x^0)$ and $R^0 = \tilde{x}^0 - x^0$
$x^1 = x^0 + \omega \cdot R^0$
for $k = 1 \dots$ do
    $\tilde{x}^k = H(x^k)$ and $R^k = \tilde{x}^k - x^k$
    $V_k = [\Delta R^0, \dots, \Delta R^{k-1}]$, $\Delta R^i = R^{i+1} - R^i$
    $W_k = [\Delta \tilde{x}^0, \dots, \Delta \tilde{x}^{k-1}]$, $\Delta \tilde{x}^i = \tilde{x}^{i+1} - \tilde{x}^i$
    decompose $V^k = QU$
    solve $UZ = Q^T$ for $Z \in \mathbb{R}^{k\times n}$
    $J_{\tilde{R}}^{-1} = J_{\tilde{R}}^{-1,(N)} + (W_k - J_{\tilde{R}}^{-1,(N)} V_k) Z$
    $\Delta\tilde{x}^k = -J_{\tilde{R}}^{-1}(\tilde{x}^k) R^k$
    $x^{k+1} = \tilde{x}^k + \Delta\tilde{x}^k$
end for
```

Discussion Anderson acceleration dates back to the original work of Donald G. Anderson from 1965 [1]. For a long time, the method was mainly applied for electronic structure computations. The chemistry community referred to the method as mixing (compare e.g. [154]), sometimes also Pulay mixing [165] or *direct inversion in the iterative subspace* (DIIS) [169]. The method regained interest recently as it shows promising features in many different applications, ranging from maximum-likelihood estimates in computational statistics [217] to dislocation dynamics [98] and groundwater flow problems [135]. In particular, multi-physics applications, where complicated models and often legacy codes need to be coupled, can benefit from the black-box nature of Anderson acceleration (see e.g. [97, 198],).

The recent new attention manifested itself in theoretical works by Fang and Saad in 2007 [78], Walker and Ni in 2011 [217], and Toth and Kelley in 2015 [196] . A minisymposium[29] at the SIAM CSE 2015 conference as well as the ICERM workshop on *Numerical Methods for Large-Scale Nonlinear Problems and Their Applications*[30] further showed the recently grown interest in Anderson Acceleration. At the

[29] http://meetings.siam.org/sess/dsp_programsess.cfm?SESSIONCODE=19874

[30] https://icerm.brown.edu/topical_workshops/tw15-5-nmlnp/

ICERM workshop, Donald Anderson himself first talked about this topic after the recent developments [2]. He reported on the re-invention of his method in various communities over the past 50 years, a fact that comes as no surprise as the least-squares idea of the method appears quite natural. In fact, as stated above, Anderson acceleration has also been known as DIIS or Pulay mixing, but also as non-linear GMRES. Finally, also the partitioned FSI community has re-invented this method, a fact that Donald Anderson was not yet aware of. I discuss the connection to the FSI methodology in the next section.

Fang and Saad distinguish between type I and type II multi-secant methods [78]. Type I methods minimize in terms of the actual Jacobian, whereas type II methods minimize in terms of the inverse Jacobian. This can be the difference between the current and a previous (inverse) Jacobian, for generalized Broyden, or the norm of the Jacobian itself, for Anderson. Walker and Ni reuse this classification [217]. Type I methods lead, thus, to approximations of the Jacobian itself. They can, however, be explicitly transformed to approximations of the inverse Jacobian, by means of the Sherman-Morrison-Woodbury formula[31]. The above, in Algorithm 1, introduced version of Anderson acceleration is identical to the type II method of Fang and Saad[32] and identical to the original formulation by Anderson if we set the mixing parameters $\beta^k > 0$ in both descriptions to 1. These mixing parameters constitute an under-relaxation between the Picard value $\tilde{x}^k$ and the accelerated value x^{k+1} and therefore "reflect the tacit assumption that the underlying Picard iteration is converging", [2]. The above introduced generalized Broyden method, however, differs from the type II description by Fang and Saad. The latter uses an update directly from x^k to x^{k+1} whereas the variant above uses an update from the Picard iterate $\tilde{x}^k$ to x^{k+1}, compare Table 4 for details. This is important, as, in general, generalized Broyden methods are sensitive to the initial (inverse) Jacobian, whose choice is in practice non-trivial. Setting it to $0^{n\times n}$, for example, renders the above type II variant to an Anderson type II method in the first timestep, while the type II method of Fang and Saad uses $-I^{n\times n}$. Marks and Luke [137] also use the Fang and Saad generalized Broyden method type II, but they use quite involved application specific heuristics to choose an effective initial inverse Jacobian. The related type I update of the above introduced variant would need the "inverse" of $0^{n\times n}$ as a favorable starting value. Such a choice is not unfeasible (compare the results in Section 3.4), but far from being robust in practical applications.

	Type I: $\|J - J^{(N)}\| \to \min$	**Type II**: $\|J^{-1} - J^{-1,(N)}\| \to \min$
$J_R X = V$	$J_R = J_R^{(N)} + (V - J_R^{(N)} X)(X^T X)^{-1} X^T$	$J_R^{-1} = J_R^{-1,(N)} + (X - J_R^{-1,(N)} V)(V^T V)^{-1} V^T$
$J_{\tilde{R}} W = V$	$J_{\tilde{R}} = J_{\tilde{R}}^{(N)} + (V - J_{\tilde{R}}^{(N)} W)(W^T W)^{-1} W^T$	$J_{\tilde{R}}^{-1} = J_{\tilde{R}}^{-1,(N)} + (W - J_{\tilde{R}}^{-1,(N)} V)(V^T V)^{-1} V^T$

Table 4: Jacobian update formulas for generalized Broyden methods. The first row corresponds to the original formulations by Fang and Saad [78], whereas the second row are the methods introduced in this work. $X_k = \left[\Delta x^0, \Delta x^1, \ldots, \Delta x^{k-1}\right]$, with $\Delta x^i = x^{i+1} - x^i$.

The interpretation of Anderson acceleration as a multi-secant method by Fang and Saad is of practical importance, but gives no new insight into the convergence of the method. Walker an Ni proof that Anderson acceleration type II is *essentially equivalent* to GMRES in the linear case, meaning that both iteration series can be constructed from each other. In the same spirit, they conclude the equivalence of Anderson type I to the Arnoldi method. Haelterman et al. show in independent work a relationship of Anderson type I to GMRES [110]. The authors refer to the method, however, as quasi-Newton least-squares method. I come back to this fact in the next section. Again in independent work, Rohwedder and Schneider also show the analogy to GMRES in the linear case, the interpretation as a secant method and, furthermore, an interpretation as a quasi-Newton method with a finite difference approximation of the Jacobian [169].

All results, those of Walker and Ni, those of Haelterman et al. and those of Rohwedder and Schneider give convergence of Anderson acceleration in the linear case for $k > N$. This case is, however, of poor practical relevance. Rohwedder and Schneider further try to deduct (heuristic) conditions for the linear dependence of the columns of V under which a superlinear convergence can be observed. Toth and Kelley give in 2015, finally, more practically relevant convergence results [196]. Anderson type II is

[31] $A \in \mathbb{R}^{n\times n}, \det(A) \neq 0, U \in \mathbb{R}^{n\times m}, V \in \mathbb{R}^{m\times n}, \det(A+UV) \neq 0 : (A+UV)^{-1} = A^{-1} + A^{-1}U(I - VA^{-1}U)^{-1}VA^{-1}$, compare for example [103].

[32] The formulation of Fang and Saad uses $J_R X = V$, combined with $\|J_R^{-1,(N))} + I\|_F \to$ min, which leads to the same type II method as the present formulation, $J_{\tilde{R}} W = V$, combined with $\|J_{\tilde{R}}^{-1,(N))}\|_F \to$ min. This does, however, not hold not true for Anderson type I. For the results of Section 3.4, I use the Anderson type I form of Fang and Saad formulation.

locally r-linearly convergent if the fixed-point map is a contraction and the coefficients stay bounded. This means, $\exists c \in [0;1[$ and $M > 0$ such that, if x^0 is close enough to the fixed-point x^*,

$$\|x^k - x^*\| \leq Mc^k \|x^0 - x^*\| \ .$$

The authors argue that the boundedness of the coefficient is no limitation in practice. The fixed-point map being a contraction, however, hinders the theoretical result to hold for most FSI cases. This would mean that the Picard iteration already converges, which is not true for most challenging FSI scenarios. In the linear case, the authors show that the results also hold in the global case (i.e. for any starting point x^0), and can even be tightened to q-linear convergence (i.e. $M = 1$). Furthermore, the authors test numerically how the substitution of the l_2-minimization by l_1 or l_{inf} influences the performance of Anderson acceleration. Both cases lead to a linear program, which comes at a higher cost than the least-squares problem, but finally at no better convergence rate.

I close this theoretical discussion with a loose comparison between the methods of Newton, Picard and Anderson. This comparison follows a similar panel discussion held at the above mentioned ICERM workshop. One important difference between Picard and Anderson is that Picard *forgets about the past immediately.* Therefore, Picard can better cope with noisy iterations, such as encountered in turbulent FSI. In absence of such noise, Anderson typically outperforms Picard, whereas the ratio between both methods highly depends on the application. For FSI, for example, Picard even tends to diverge such that Anderson can be viewed as a stabilization technique. On the other hand, if an analytical Jacobian is available, Newton tends to perform better in terms of iteration counts than Anderson. This does typically not hold true, if the Jacobian stems from a finite difference approximation. [135, 217] give comparisons of Anderson and Newton. For FSI, a clean comparison is hard to establish and rarely studied, compare the discussion in Section 1.2. The general believe of the community is that an exact Newton method outperforms a quasi-Newton method, but not by orders of magnitude. An exact Newton is, however, not available for black-box solvers and cumbersome to construct for all other solvers. If we boil the discussion down to a pure local convergence order, Newton would beat Anderson two to one. Further away from the fixed-point Newton shows, however, only linear convergence (or even divergence), just as Anderson does. To do a fair theoretical comparison, a non-asymptotic convergence theory might be of help, such as discussed by Matthew Knepley at several occasions[33].

3.3 Quasi-Newton Coupling Schemes for Fluid-Structure Interaction

This section combines the fixed-point systems of Section 3.1 and the fixed-point equation solvers of Section 3.2 to construct concrete coupling algorithms for FSI. Along with this construction, the link between the FSI coupling schemes and the fixed-point acceleration techniques is established. To this end, well-known quasi-Newton coupling approaches from the literature are sorted in these categories. Figure 26 visualizes the recent history of this link. For a broader review of FSI black-box coupling schemes, the reader may refer to, for example, [99].

Section 3.1 introduced four coupling systems, which result together with the four solvers of Section 3.2 (Anderson acceleration and generalized Broyden, both in type I and II) in 16 combinations:

$$\{(\text{GS}), (\text{J}), (\text{SP}), (\text{B})\} \times \{\text{AA-I}, \text{AA-II}, \text{GB-I}, \text{GB-II}\} \ .$$

Please remember: all type I methods can be reformulated as approximation to the inverse Jacobian by means of the Sherman-Morrison-Woodbury formula.

Historically, the first described method in this spirit is the interface-Newton-Krylov method or the interface-GMRES method, respectively by Michler, van Brummelen and van Borst from 2004 [144, 146, 207], formulated for (GS). This method marks an important milestone in the development of FSI coupling schemes, though it is of no practical relevance any longer as the method is clearly outperformed by other variants (e.g. [63]) and therefore also not covered in Section 3.2. The block-iterative system (B) cannot be combined with any arbitrary fixed-point solver as the system requires a block-iterative solution by definition. Otherwise, the system would fall back to (J). For further details, the reader may consult Section 4.4 of [176]. Vierendeels et al. use (B) to formulate the interface block quasi-Newton least-squares (IBQN-LS) coupling scheme in 2007 [216]. This method corresponds to Anderson type I.

[33] To my best knowledge, such work has not yet been published.

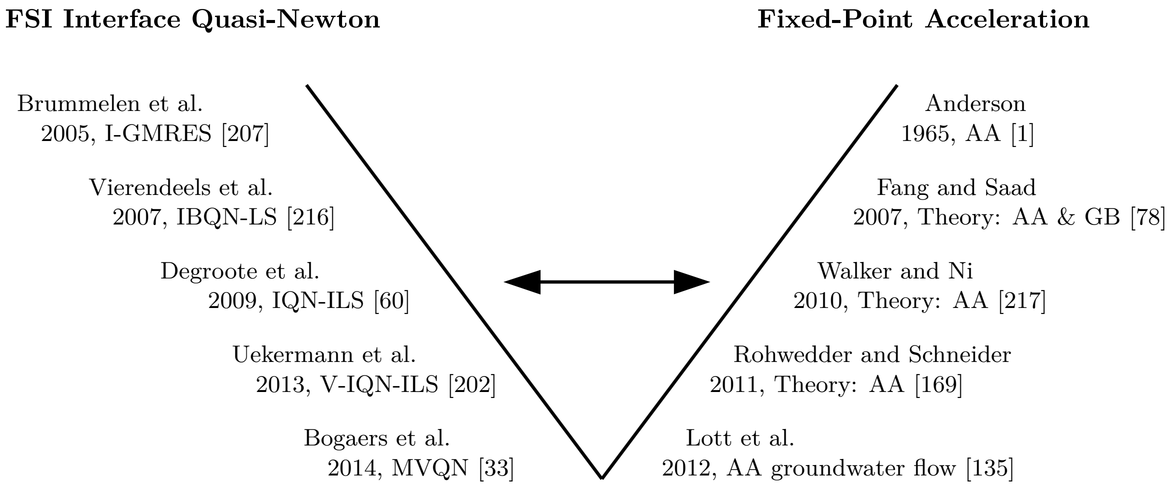


Figure 26: Developments during the last decade and literature excerpt of the partitioned FSI community compared to the fixed-point acceleration community.

Recently, in 2014, Bogaers et al. formulate the multi-vector quasi-Newton (MVQN) scheme [33], which corresponds to generalized Broyden type I for (B). In 2009, Degroote et al. describe the interface quasi-Newton inverse least-squares (IQN-ILS) method [60], which corresponds to Anderson type II for (GS). In the same spirit, Haelterman et al. study the quasi-Newton least-squares method (QNLS) [110], the Anderson type I variant. Minami and Yoshimura solve (GS) by means of a line extrapolation technique respectively by a classical Broyden scheme [148]. The results of [176] suggest that such simple schemes are outperformed by the more advanced schemes from Section 3.2. (SP) is already studied in the literature, e.g., [66, 127], but not yet in a clean black-box manner.

Various performance comparisons between the different approaches exist in literature. [63] shows a similar performance of IQN-ILS compared to IBQN-LS, with the advantage of IQN-ILS of being easier to implement. Both methods outperform Aitken underrelaxation and Interface-GMRES. Depending on the testcase, the difference can be a slight one or a tremendous one. [206] gives theoretical insight into the difference between Aitken underrelaxation and I-GMRES by means of a simple testcase. Aitken underrelexation can perform well if the initial error in every timestep is dominated by only a few smooth modes. [60, 215] show that the spatial low-frequent modes dominate in terms of the added-mass effect. Whereas Aitken underrelaxation tries to stabilize all modes uniformly, sophisticated methods like IQN-ILS adapt better to the unstable modes. [33] shows that MVQN performs similarly as IBQN-LS without the need to tune the number of reused timesteps. Both methods outperform Aitken underrelaxation and a simple Broyden method.

The contribution of this work consists in generalizing the IQN-ILS method, i.e., Anderson type II, to (J) and (SP), to obtain parallel coupling schemes. Preliminary results are already published in [202] and extensively studied in [142]. Please note that these publications denote (J) as the vectorial system and (SP) as the parallel system. Furthermore, this work introduces generalized Broyden coupling variants for (GS), and similarly for (J) and (SP). These results are part of the master thesis of Klaudius Scheufele [176] and already published in [129]. A method similar to the generalized Broyden scheme for (GS) was also introduced in independent work by Rob Haelterman [108]. Table 5 gives an overview on the FSI coupling schemes, their original names and main reference, and their connection to the multi-secant methods discussed in the last section.

Various authors discuss the application of these FSI coupling schemes in a multi-level context [29, 65, 178, 209, 240]. While classical approaches are based on (GS), this is no necessity. In joint work, we show the possibility to apply a multi-level approach to (J) [28].

3.4 Basic Numerical Tests: 1D Tube

This section studies a simple, yet very standard FSI testcase: the flow through an elastic tube is formulated by means of a 1D model. This testcase is certainly of no practical meaning, but it fully

	(GS)	(J)	(SP)	(B)
AA I	QNLS [110]	V-IQN-LSHJ* [176]	SP-AA1*	IBQN-LS [216]
AA II	IQN-ILS [60]	V-IQN* [202]	P-IQN* [202]	-
GB I	S-IQN-MVJ* [176]	V-IQN-MVJ* [176]	SP-GB1*	MVQN [33]
GB II	S-IQN-IMVJ* [176]	V-IQN-IMVJ* [176]	SP-GB2*	-

Table 5: Quasi-Newton coupling schemes for fluid-structure interaction: *original* names and main reference. Legend: * Methods that mark a contribution of this work. - Combinations that are not applicable.

captures the added-mass instability and, thus, allows to study coupling schemes by simple MATLAB code. The testcase is for example used in [62] and the results of this chapter are partly already published in [202, 142, 176]. The purpose of this section is to compare and overview the complete zoo of methods of Section 3.3 and draw conclusion on the practical relevance of each one of them. This leads, in particular, to the decision which methods are supported by preCICE.

3.4.1 Testcase Description

Analytical Description The scenario consists of an incompressible and inviscid flow through a flexible tube. The flow is assumed to be radial-symmetric. Averaging over the radial coordinate leads to a 1D model. Compare Figure 27 for a schematic drawing of the scenario. Conservation of momentum and mass reads

$$\begin{aligned} \partial_t(au) + \partial_x(au^2) + a\partial_x p &= 0\ , \qquad (4) \\ \partial_t a + \partial_x(au) &= 0\ , \qquad (5) \end{aligned}$$

where u denotes the flow velocity in axial direction, p the kinematic pressure, and a the cross section area of the tube. A time-varying inlet velocity and a non-reflecting outlet condition are imposed:

$$u_{in} = u_0 - \frac{u_0}{100}\sin^2(\pi\frac{t}{T}) \text{ and } \partial_t u = \frac{1}{c}\partial_t p\ ,$$

with the wave speed $c^2 := a/d_p a = c_{mk}^2 - \frac{p}{2}$, the Moens-Korteweg wave speed $c_{mk} = \sqrt{Eh/2\rho r_0}$ and Young's modulus E . The elastic wall is modeled by a Hookean constitutive law. The inertia of the tube wall is, therefore, neglected, which leads to a significant added-mass effect. As the inviscid fluid exerts only stress in circumferential direction on the structure, this leads to a pure radial motion of the tube wall. The cross section area becomes, thence, an explicit function of the pressure:

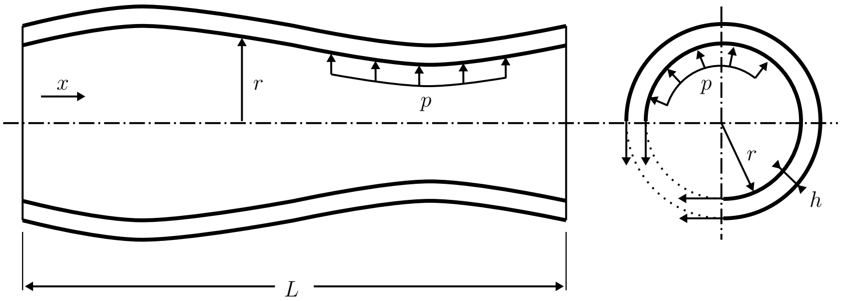


Figure 27: 1D flow through a flexible tube: schematic drawing of the deformed tube. Left: cut along the tube. Right: lateral cut through the tube. The test scenario is based on [62]. Picture taken from [99].

$$a = a(p) = a_0 \left(\frac{p_0 - 2c_{mk}^2}{p - 2c_{mk}^2} \right)^2 ,$$

where a_0 and p_0 denote fixed reference values. For further details on the modeling of this scenario, the reader may refer to [62]. Please note, that, due to the 1D nature of the scenario, the coupling variables f and d are substituted by the pressure p and the cross section area a.

Discretization The computational domain of length L is discretized with n equidistant cells of length $\Delta x = L/n$. The variables u_i, p_i and a_i are defined at the cell centers. The grid of the fluid domain coincides with the structure grid and the coupling interface. Equations (4) and (5) are discretized by a second-order central finite volume scheme, combined with a first order upwind scheme for the convective term. A pressure stabilization with coefficient

$$\alpha = \frac{a_0}{u_0 + \frac{\Delta x}{\Delta t}}$$

is applied. For time discretization, a backward Euler scheme with fixed timestep size Δt is used. The discretized system reads: $i = 1, \ldots, N-1$

$$\begin{aligned}\frac{\Delta x}{\Delta t}\left(u_i^{(N+1)}a_i^{(N+1)} - u_i^{(N)}a_i^{(N)}\right) + \Big[u_i u_{i+\frac{1}{2}} a_{i+\frac{1}{2}} - u_{i-1} u_{i-\frac{1}{2}} a_{i-\frac{1}{2}}\Big]^{(N+1)} & \\ + \frac{1}{2}\Big[a_{i+\frac{1}{2}}(p_{i+1} - p_i) + a_{i-\frac{1}{2}}(p_i - p_{i-1})\Big]^{(N+1)} &= 0\ , \\ \frac{\Delta x}{\Delta t}\left(a_i^{(N+1)} - a_i^{(N)}\right) + \Big[u_{i+\frac{1}{2}} a_{i+\frac{1}{2}} - u_{i-\frac{1}{2}} a_{i-\frac{1}{2}}\Big]^{(N+1)} - \alpha\Big[p_{i-1} - 2p_i + p_{i+1}\Big]^{(N+1)} &= 0\ .\end{aligned}$$

The subscript $i \pm 1/2$ denotes values at the cell interfaces, e.g.

$$u_{i-\frac{1}{2}} = \frac{1}{2} \cdot (u_{i-1} + u_i)\ .$$

The superscripts (N) refers to the time level $t^{(N)} = N \cdot \Delta t$. For the boundary conditions, the applied linear extrapolation of the inlet pressure and the outlet velocity reads

$$p_{in} = 2p_1 - p_2 \quad \text{and} \quad u_{out} = 2u_n - u_{n-1}\ ,$$

whereas the pressure-outlet condition is discretized as

$$p_{out} = 2\left[c_{mk}^2 - \left(\sqrt{c_{mk}^2 - \frac{p_{out}^n}{2}} - \frac{u_{out} - u_{out}^n}{4}\right)^2\right]\ .$$

Initially, velocity, pressure, and cross section area are set to their reference values u_0, p_0, and a_0. Similar to [62], I define the dimensionless structural stiffness

$$\kappa = \frac{\sqrt{\frac{Eh}{2\rho r_0} - \frac{p_0}{2}}}{u_0}$$

and the dimensionless timestep size

$$\tau = \frac{u_0 \Delta t}{L}$$

as parameters of the scenario. The stability analyses of [62] shows that the added-mass effect grows in significance for decreasing κ and decreasing τ. n is fixed to 100, whereas a full period of the inlet velocity $[0; T]$ is simulated with 100 timesteps.

3.4.2 Results and Conclusions

This subsection gives an overview of numerical results for (GS), (J), and (SP), solved by either a pure Picard iteration, an Aitken underrelexation, an Anderson acceleration or a generalized Broyden method. For the latter two, both types I and II are analyzed, whereas type I is based on an explicit linear system solve, i.e. not transformed by means of the Sherman-Morrison-Woodbury formula. I do not analyze any results for (B) as the focus of this chapter is on parallel coupling schemes and (B) implies a sequential scheme. Furthermore, [63] showed that schemes based on (B) show similar performance as schemes based on (GS), whereas the latter clearly simplifies the implementation. The interested reader is, once again, referred to [176] for more comparisons. For this thesis, (GS) serves as the sequential baseline. I use the notation J-AA2, to refer to, for example, an Anderson acceleration type II for (J). If columns from previous timesteps get reused, I add the number of timesteps in brackets, e.g., J-AA2(5) for 5 reused timesteps. The 1D elastic tube is parametrized by $\kappa \in \{10, 100, 1000\}$ and $\tau \in \{0.001, 0.01, 0.1\}$, similar to, e.g., [62]. Figure 28 shows the simple physical results of the scenario over various timesteps. All converged coupling algorithms result in similar physical results up to an absolute l_2 error of 10^{-7} with respect to the cross section area.

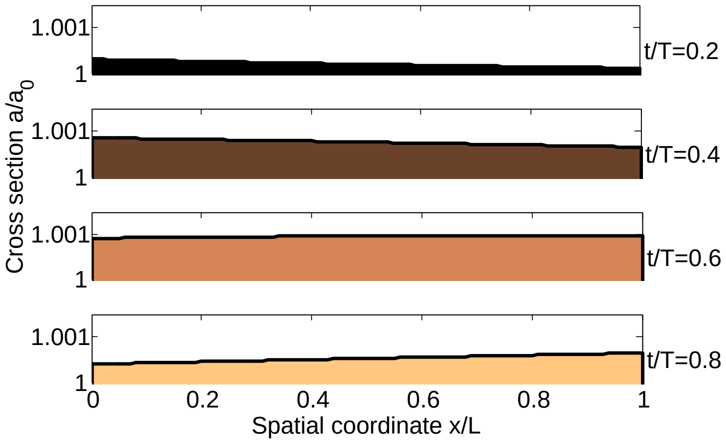


Figure 28: Cross section area values a/a_0 for different timesteps for $\tau = 0.01$ and $\kappa = 10$. Picture taken from [202].

Numerical Settings The numerical results are obtained by means of a simple MATLAB implementation. The fluid solver uses a Newton iteration as a non-linear solver up to a relative error in the residual of 10^{-16} or 50 iterations. The linearized system is solved by a direct solve. The fix-point iterations in every timestep start with the converged values from the previous timestep as initial guesses (constant extrapolation). I use a relative convergence criterion of 10^{-7} for both, cross section area and pressure, compare the discussion in Section 3.1. If convergence is not achieved in 100 iterations, I treat a run as diverged. Aitken underrelexation and all quasi-Newton methods start with an underrelexation of $\omega = 0.1$ in the first timestep. Aitken underrelaxation starts, further, each other timestep with $\omega = \min(\omega_{old}, 0.5)$, whereas ω_{old} refers to the last relaxation parameter of the previous timestep. Quasi-Newton methods use all columns of the previous timestep as the search space of the first iteration in every timestep, even if no reuse is specified. Always, all columns of the current timesteps are considered. Additionally, columns from previous timesteps are added if specified. For reused timesteps with more than 5 columns, only the first 5 are added (i.e., the oldest 5). Furthermore, to avoid a nearly quadratic system, the oldest columns are dropped if the total reused columns exceeds $N/2$. Section 3.6.2 gives more information on how to formalize such heuristics. I use the built-in MATLAB functions `qr` and `linsolve` for the QR-decompostion and the linear solve, respectively. Only Anderson acceleration type II uses a QR-decompostion. All other quasi-Newton scheme use a direct linear solve instead. For the generalized Broyden schemes, I use the initial Jacobian $0^{n\times n}$ and the initial inverse Jacobian $10^9 \cdot I^{n\times n}$.

Simple Schemes Table 6 lists average iteration numbers for a plain Picard iteration and an Aitken underrelexation, applied to all three coupling systems. Picard diverges for the more challenging scenarios, whereas Aitken shows mostly stable results. For both methods, (GS) and (SP) show very similar results, while (J) leads, with Picard, to roughly twice as many iteration than (GS). This comes at no surprise as a simple Picard iteration applied to (J) leads to two independent chains of (GS) systems [142]. Figure 29, left, illustrates the corresponding residual zig-zag behavior. However, even Aitken does not succeed in connecting these two chains, but deteriorates the performance. I conclude that simple Picard-based schemes are no option for (J). The results further below show, furthermore, that Aitken is clearly outperformed by the quasi-Newton schemes, up to a factor of roughly 8 for the more challenging scenarios.

Quasi-Newton Schemes Table 7 lists the mean iteration numbers for Anderson acceleration and generalized Broyden, in both types, for all three coupling systems. All schemes, with the exception of GB1, show a robust performance. GB1 tends to behave sensitively to the initial Jacobian. The fixed choice of $10^9 \cdot I^{n\times n}$ leads to divergence for certain parameter settings, which furthermore depend on the applied coupling system: GS-GB1 diverges for the easier $\kappa = 1000$, whereas J-GB1 diverges for a higher added-mass effect. AA1 and AA2 show very similar performance, whereas GB2 outperforms GB1 persistently. In general, GB2 also outperforms AA1 and AA2. This is due to the implicit capturing of past timesteps. AA1 and AA2 can make up for this by explicitly reusing columns from past timesteps

GS-Picard				J-Picard				SP-Picard			
$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10
0.1	4.27	7.23	div	0.1	9.00	15.07	div	0.1	4.26	7.23	div
0.01	6.53	div	div	0.01	14.75	div	div	0.01	6.53	div	div
0.001	44.51	div	div	0.001	101.46	div	div	0.001	44.62	div	div

GS-Aitken				J-Aitken				SP-Aitken			
$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10
0.1	4.01	5.03	10.04	0.1	16.71	19.49	54.45	0.1	4.01	5.03	10.06
0.01	4.99	8.11	26.32	0.01	18.38	42.08	div	0.01	4.99	8.11	25.99
0.001	7.68	23.15	div	0.001	38.07	div	div	0.001	7.68	23.23	div

Table 6: Mean iterations per timestep for the 1D elastic tube. Various coupling systems are compared for a pure Picard iteration as well as an Aitken underrelexation.

as the results further below illustrate. Just like for the simple schemes above, (GS) and (SP) show very similar results. (J) shows a worse performance than (GS), but not by a factor of 2, like for the simple schemes. Thus, the quasi-Newton schemes manage to connect the two chains of (J).

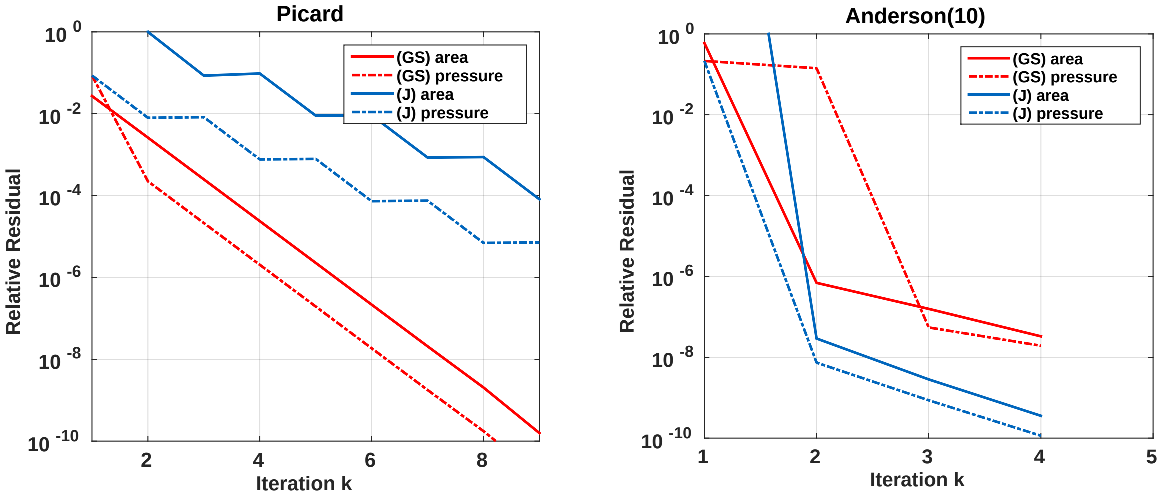


Figure 29: Relative residual for the 1D elastic tube at timestep 20. Left: Picard iteration for $\tau = 0.1$ and $\kappa = 100$. Right: AA2(10) for $\tau = 0.01$ and $\kappa = 10$.

Reuse of Past Information Figure 30 illustrates how the reuse of past information influences the performance of AA2 and GB2. For GB2, information from past timesteps is already implicitly captured by the Jacobian update. Thus, there is no huge gain visible if columns are also reused explicitly. Still, the reuse of 1 or 2 timesteps might be beneficial. Also, for higher values the performance is pretty stable, especially for (J). This renders a tuning for the optimal reuse value unnecessary. The performance of AA2, however, depends highly on this parameter. For a higher added-mass effect (here a smaller τ), also the optimal parameter is higher. Furthermore, (J) shows a higher optimal parameter than (GS). Table 8 summarizes the optimal values for each method and its performance. For this optimal configuration, GB2 and AA2 show similar performance. In general, all three coupling systems (GS), (J), and (SP) also show similar performance then. The parallel (J) even outperforms the serial (GS) in the more difficult cases as the first one often converges in 2 iterations, whereas (GS) needs 3. This difference stem from the fact that, for example, J-AA2 reuses past information from both, a and p. GS-AA2 only uses past information from a, which reaches then the p convergence only in the second iteration. Finally, the convergence criterion sees this behavior naturally shifted by one iteration. Figure 29, right, illustrates this difference at an arbitrary timestep.

GS-AA1				**J-AA1**				**SP-AA1**			
$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10
0.1	2.99	3.08	4.29	0.1	4.01	5.14	8.68	0.1	2.99	3.09	4.31
0.01	3.09	3.74	8.20	0.01	4.68	7.17	16.56	0.01	3.05	3.79	8.20
0.001	3.83	7.67	div	0.001	7.00	15.17	39.46	0.001	3.89	7.63	22.06

GS-AA2				**J-AA2**				**SP-AA2**			
$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10
0.1	2.99	3.09	4.27	0.1	4.97	5.73	9.23	0.1	2.99	3.09	4.25
0.01	3.09	3.74	7.90	0.01	5.34	7.93	17.01	0.01	3.08	3.71	7.96
0.001	3.81	7.48	22.94	0.001	7.62	15.46	40.45	0.001	3.82	7.53	22.36

GS-GB1				**J-GB1**				**SP-GB1**			
$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10
0.1	div	3.23	4.15	0.1	3.97	4.51	9.25	0.1	3.00	3.30	4.24
0.01	div	4.24	6.36	0.01	4.68	6.40	div	0.01	3.22	4.11	6.37
0.001	div	6.90	div	0.001	6.42	div	div	0.001	4.40	6.71	div

GS-GB2				**J-GB2**				**SP-GB2**			
$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10
0.1	3.02	3.07	3.77	0.1	3.26	3.54	5.49	0.1	3.01	3.11	3.85
0.01	3.06	3.40	4.92	0.01	3.70	4.36	6.85	0.01	3.09	3.42	5.00
0.001	3.43	5.30	10.58	0.001	4.10	6.34	9.15	0.001	3.24	5.37	8.70

Table 7: Average iterations per timestep for the 1D elastic tube. Various coupling systems are compared for Anderson acceleration and generalized Broyden, both for a type I and type II update.

GS-AA2(R)				**GS-GB2(R)**			
$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10
0.1	2.99(0)	3.06(3)	3.35(2)	0.1	3.00(1)	3.07(0)	3.37(2)
0.01	3.02(4)	3.10(2)	3.87(4)	0.01	3.02(3)	3.22(1)	3.73(3)
0.001	3.16(5)	3.41(4)	8.43(5)	0.001	3.43(0)	3.66(2)	6.15(6)

J-AA2(R)				**J-GB2(R)**			
$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10
0.1	2.75(3)	3.74(2)	4.04(3)	0.1	3.26(0)	3.05(2)	3.87(5)
0.01	2.31(2)	3.37(3)	3.55(8)	0.01	3.07(3)	3.41(3)	3.39(5)
0.001	2.69(3)	3.22(8)	11.54(8)	0.001	2.50(2)	2.95(4)	4.92(6)

SP-AA2(R)				**SP-GB2(R)**			
$\tau\backslash\kappa$	1000	100	10	$\tau\backslash\kappa$	1000	100	10
0.1	2.99(0)	3.09(0)	3.48(2)	0.1	3.01(0)	3.11(0)	3.43(2)
0.01	3.04(4)	3.10(2)	3.75(6)	0.01	3.03(4)	3.42(0)	3.62(4)
0.001	3.16(5)	3.31(7)	9.40(4)	0.001	3.24(0)	3.48(2)	5.51(7)

Table 8: Optimal choices (in brackets) for the number of reused timesteps and their performances in terms of average iterations per timestep for the 1D elastic tube. Type II methods for Anderson acceleration and generalized Broyden with various coupling systems are compared.

Conclusion I conclude from these results to implement in preCICE and further study only GS-AA2, J-AA2, GS-GB2, and J-GB2. First, (SP) shows no drastic differences in performance to (J) if compared for optimal methods. The implementation of (SP) would, however, be much more involved than the more natural choice (J) as it would require a new adapter for the structure solver due to the different choices of input and output, assumed that (GS) is already implemented. Second, I restrict the further study to type II methods. Type I methods showed robustness issues (for generalized Broyden) and also mediocre performance compared to type II methods, though Broyden originally called type I the *good* and type

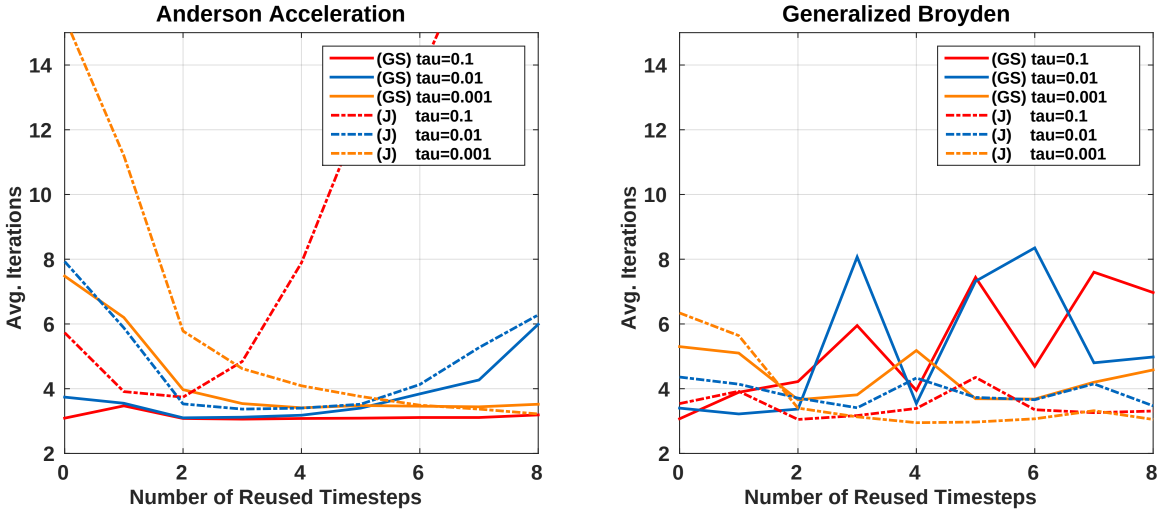


Figure 30: Average iterations per timestep for the 1D elastic tube and different number of reused timesteps. AA2 (left) performance is compared against GB2 (right) performance for (GS) and (J), and different values for τ. $\kappa = 100$.

II the *bad* update [35]. Finally, there is a valid desire to have both options, Anderson acceleration type II and generalized Broyden type II, available for practical applications as both have significantly different properties, in spite of the similar performance. On the one hand, generalized Broyden renders the tuning of reused information unnecessary. On the other hand, Anderson acceleration allows for a matrix-free implementation. This makes the method not only more memory efficient, but also simplifies the parallelization drastically as studied in Section 4.4.

The next section elaborates the implementation of the four methods of choice in preCICE. Afterwards, Section 3.6 discusses advanced numerical details. Finally, Section 3.7 gives detailed performance results of the four coupling schemes applied to 2D and 3D testcases.

3.5 Implementation in preCICE

This section gives insights into the implementation of the aforementioned coupling schemes in preCICE. I focus on the general workflow of the coupling schemes, which are independent of whether preCICE is executed in serial, on a server or on distributed data. Section 4.4 gives more information on how the underlying numerical operations are ported to distributed data. Originally, the only supported quasi-Newton scheme in preCICE was Anderson acceleration (type II) applied to (GS) [99]. In the following, I explain the generalization to (J) and also to generalized Broyden (type II).

The Package `cplscheme` As detailed in Section 2.2, preCICE consists of several packages. The purpose of the package `cplscheme` is to steer the data exchange and, thus, the synchronization between two or more participants. This covers also implicit coupling schemes, where a sub-iteration between multiple participants is performed in every timestep. Accordingly, `cplscheme` offers methods for measuring the convergence and accelerating the fixed-point iterations. The latter is referred to as post-processing in the implementation. Furthermore, `cplscheme` offers methods to deal with non-matching time discretizations, including sub-cycling. Figure 31 lists the API of `cplscheme`, which follows closely the API of preCICE itself. For more details, the interested reader is referred to [99].

```
void initialize();
void initializeData();
void advance ();
void finalize();
bool isCouplingOngoing();

bool isActionRequired(std::string& actionName);
void performedAction(std::string& actionName);
void requireAction(std::string& actionName);
```

Figure 31: Interface of the `cplschem` package (excerpt). The design follows the main preCICE interface very closely (compare Section 2.1). The interface provides methods to control the workflow as well as methods to pass actions from the main preCICE interface to `cplscheme`. Further methods to control the timestep size and absolute time are suppressed in this listing.

Software Architecture Figure 32 sketches the class hierarchy of `cplscheme`. I distinguish original parts, already implemented as part of [99], and recent work by the colors blue and orange, respectively. The overall goal of the newly implemented parts is to allow for a flexible combination of any coupling system with any post-processing method. A modular programming, which clearly separates the coupling systems, `SerialCouplingScheme` or (GS) and `ParallelCouplingScheme` or (J), from the post-processing methods, achieves this goal. Arbitrary combinations are possible.

To allow the coupling of more than two participants, two different concepts are implemented. The `CompositionalCouplingScheme`, which uses a composite design pattern and the `MultiCouplingScheme`. For the first, [99] gives more information on the implementation. For the latter, one participant serves as a central instance computing the post-processing and instantiates `MultiCouplingScheme`, whereas all other participants instantiate `ParallelCouplingScheme`. Section 3.8 gives detailed information on the numerical background of both methods. The actual numerical computations of the quasi-Newton schemes are outsourced to the classes `QRFactorization` and `ParallelMatrixOperation`. Section 4.4 gives more details. A further new component of this work is `Preconditioner`, which is discussed numerically in Section 3.6.3.

Steering of Coupling Schemes Figure 33 puts the differences between `SerialCouplingScheme` and `ParallelCouplingScheme` in a nutshell. I only consider the difference for an implicit coupling since an explicit coupling only constitutes a subset of the functionality. The original `SerialCouplingScheme` distinguishes between a first participant F and a second participant S. While both participants start simultaneously, S is first stopped in `initialize` to receive data that F sends at the beginning of

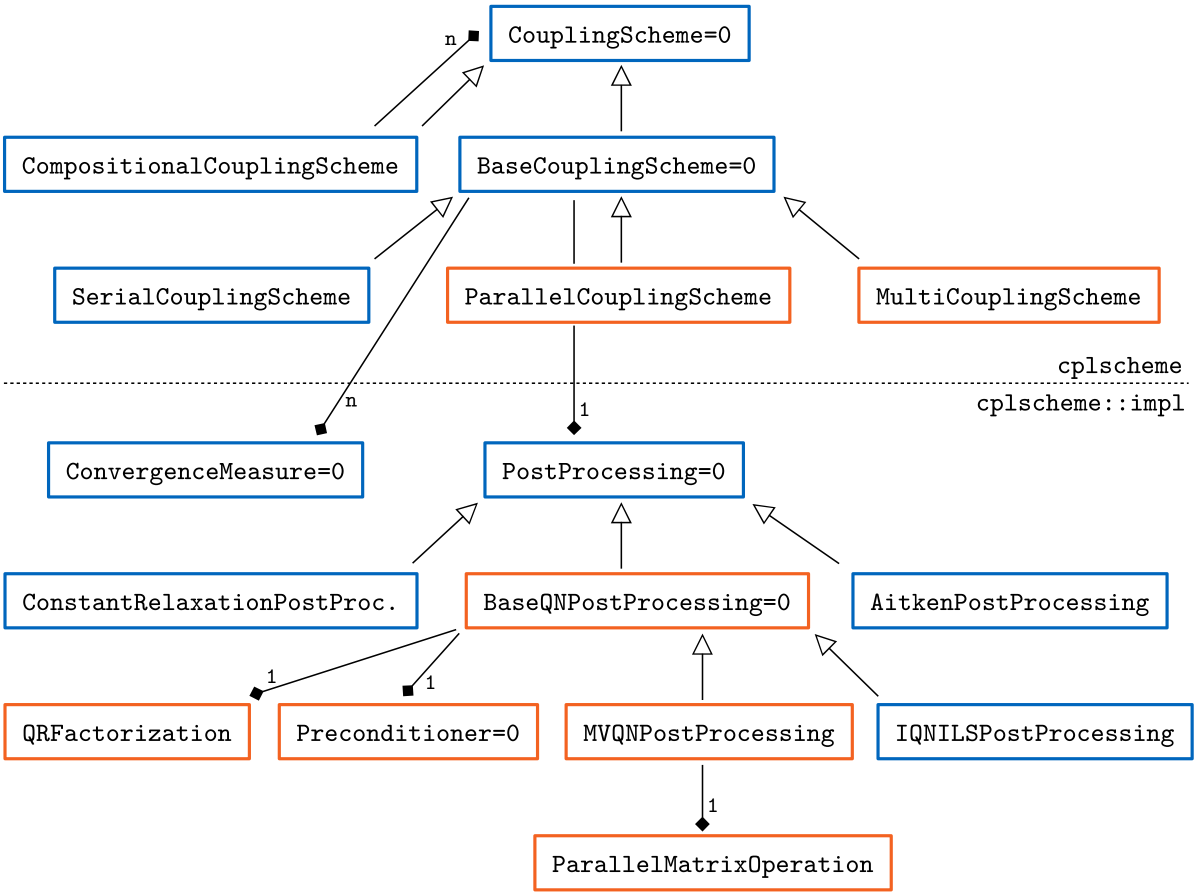


Figure 32: Software architecture of the `cplscheme` package. The arrow notation follows the UML standard (empty triangles: inheritence, filled diamonds: containement). The coupling systems are well separated from fixed-point solvers to allow for modular combinations. The interface of the package is located in the namespace `cplscheme`, whereas the internal implementations are wrapped by the sub-namespace `impl`. Original components [99] are marked in blue, whereas new components are marked in orange (post processing parts are joint work with Klaudius Scheufele).

`advance`. This leads to a staggered execution order of F and S. During `advance`, S first measures convergence of its newly computed data, followed by a post-processing step in case of non-convergence. The post-processed data is then sent to F and a receive finally awaits new data from F.

The `ParallelCouplingScheme` breaks with this asymmetry. Still, we need to distinguish between F and S to identify which participant computes the post-processing, but this classification does not longer imply a staggered execution order. Both participants compute new data simultaneously and then enter `advance`. Now F directly sends data to S as the latter needs the newest data from both participants for the post-processing. Finally, S sends the post-processed data back to F.

A further difference between the serial and parallel coupling scheme lies in the treatment of initial data. Initial data, in general, is necessary if the coupling starts from non-zero values, for example for a fixed initial displacement of the structure or during a restart. In the original serial formulation, only S can initialize data and send it to F. This is sufficient as S itself gets new data from the first iteration of F already in `initialize`. In the parallel formulation, however, setting initial data in both directions is possible. For both cases, serial and parallel, the internal flags `_hasToSendInitData` and `_hasToReceiveInitData` store whether initial data exchange is necessary or not. The user provides this information in the configuration. Afterwards, this information is passed back to the user by means of the action `WriteInitialData`. Furthermore, preCICE uses the flags to steer the initial data exchanges.

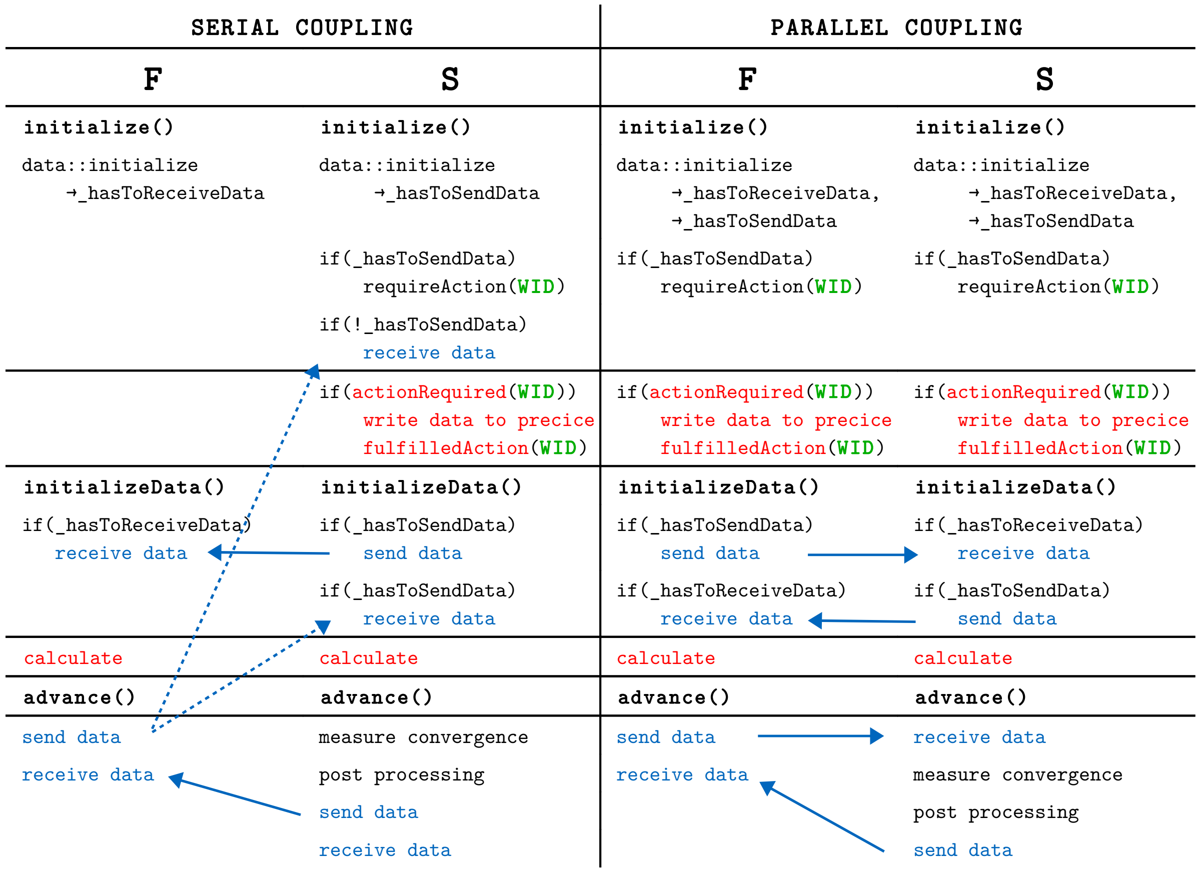


Figure 33: Differences between serial and parallel coupling in a nutshell. F and S refer to the first and the second participant, respectively. Code parts that belong to the solver itself are marked in red. Data exchange is marked in blue. WID stands for `WriteInitialData`.

3.6 Advanced Topics

To bridge the gap between the simple 1D testcase discussed in Section 3.4 and real 2D or 3D testcases, such as those discussed in Section 3.7, further numerical details need to be taken care of. This section studies those details and makes choices upon efficient, but robust settings for the further 2D/3D testcases in Section 3.7. To this end, I introduce a demanding, but simple testcase in Section 3.6.1. Section 3.6.2 uses this testcase to study filtering techniques as well as general guidelines for the QR-decomposition, which constitutes the numerical kernel of any quasi-Newton method. Afterwards, Section 3.6.3 studies various preconditioners, which are necessary for the Jacobi system (J) to deal with the different scales of displacements and forces.

3.6.1 A Numerically Demanding Testcase

Two main characteristics determine this testcase. First, it constitutes a demanding numerical regime. This implies a strong added-mass effect and a non-stationary time evolution. I do not aim to construct a testcase in a realistic parameter domain, but a testcase that clearly shows the influence of various numerical settings. Second, the testcase is very cheap to compute, allowing significant parameter studies on a simple workstation. Figure 34 shows the geometry of the testcase: a wall-mounted elastic flap in an incompressible channel flow. On the left boundary, I impose a time-varying inflow condition

$$u_{in} = u_0(1 - \cos(2\pi \cdot t \cdot 10)) \text{ and } v_{in} = 0 \; ,$$

where u_{in} and v_{in} are the inflow velocity in x and y direction. On the right boundary, an outflow condition is prescribed and no-slip walls on all other boundaries. Figure 34 further shows two different mesh configurations, a regular mesh and an adaptive mesh with 22 and 85 vertices at the coupling

fluid density	ρ_F	$1.0\,\mathrm{kg/m^3}$
dynamic viscosity	μ	$0.1\,\mathrm{kg/(m\,s)}$
reference velocity	u_0	$1.0\,\mathrm{m/s}$
structure density	ρ_S	$0.1\,\mathrm{kg/m^3}$
Young's modulus	E	$3 \times 10^4\,\mathrm{N/m^2}$
Poisson ratio	ν	0.3
timestep size	Δt	$1 \times 10^{-3}\,\mathrm{s}$

Table 9: Wall-mounted flap testcase: baseline setting for the physical parameters.

interface, respectively. Table 9 list the baseline physical parameters of the scenario. To establish robust initial conditions, the scenario is computed with a fixed geometry over a complete inflow period and restarted afterwards. For all experiments in this section, I use Anderson acceleration (type II) or generalized Broyden (type II) with reused columns from 5 and 2 timesteps, respectively, both bounded by a maximum of 30 columns for the matrices V and W, compare Section 3.2. The initial relaxation is 0.1 and an extrapolation of order 2 is used as initial guess in every iteration. The relative convergence limit for both, forces and displacement, is 10^{-4}. I use Alya Nastin and Alya Solidz as single-physics solvers. Both solvers use an inner relative convergence criterion of 10^{-6}. Figure 35 shows the physical results at various points in time during the first actual inflow period.

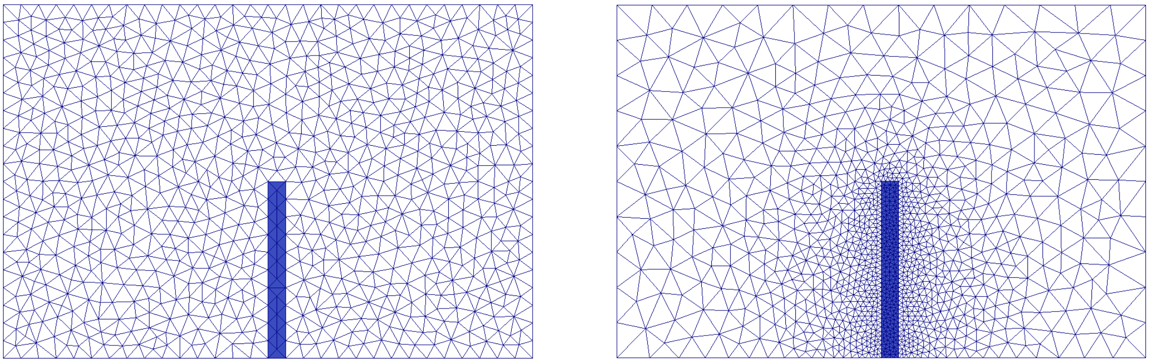


Figure 34: Meshes for the wall-mounted flap testcase: regular mesh M1 (left) and adaptive mesh M2 (right). M1 has 22 vertices at the matching coupling interface whereas M2 has 85. The domain size is $3.0\,\mathrm{m} \times 2.0\,\mathrm{m}$.

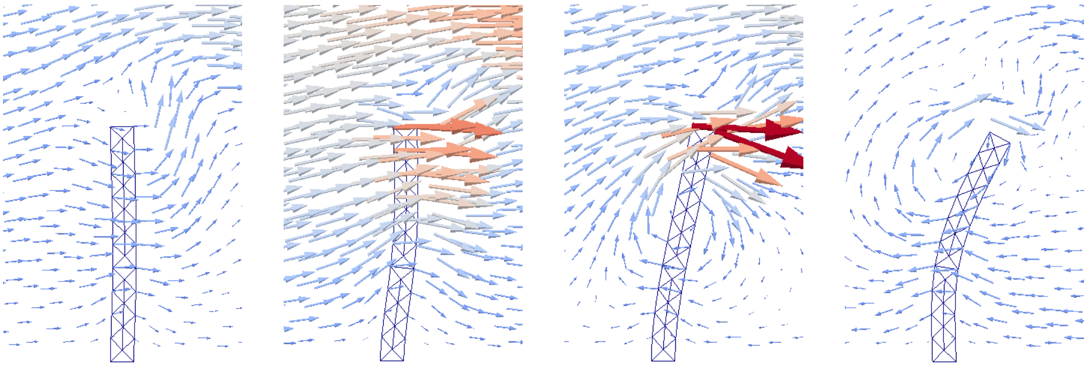


Figure 35: Physical results for the wall-mounted flap testcase (mesh M1): velocity glyphs and structural displacement at time instances $t = 0.025s$, $t = 0.05s$, $t = 0.075s$, and $t = 0.1s$ (from left to right). Glyphs are colored according to their magnitude.

3.6.2 Improving the Robustness of Quasi-Newton Schemes: Filtering

The numerical kernel of both quasi-Newton methods, Anderson acceleration and generalized Broyden, is a QR-decomposition of V (compare Section 3.2). In some cases, the columns of V can become almost linearly dependent and V, thus, almost singular, which renders the QR-decomposition unstable. This happens quite naturally during the last convergent iterations of one timestep, or if columns from previous timesteps are reused and the complete FSI scenario converges towards a stationary solution. Also, a stagnating iteration or simply too many reused columns entail risks. Filtering describes the concept of deleting columns from V and, consistently, from W to retain a good condition of V improving the robustness of the quasi-Newton schemes.

In a common effort[34] [109], we studied and compared three methods, denoted as QR1, QR2, and POD, to establish such a filtering. QR1 estimates the condition of R, which is equal to the condition of V, by comparing the diagonal elements of R to the complete norm of R. More precisely, after a complete QR-decomposition, the oldest column i of V is deleted for which

$$R_{ii} < \epsilon \cdot \|R\|_F \ .$$

In this case, the QR-decomposition is discarded and started over for the reduced V. QR2 is based on a column-wise comparison, which allows to discard the QR-decomposition already during its construction. Algorithm 2 details the filtered modified Gram-Schmidt orthogonalization. Finally, POD is based on a proper orthogonal decomposition of $V^T V$. As this method is numerically much more involved (including parallelization) and shows no clear performance benefit over QR1 and QR2 [109], we decided not to implement this method in preCICE. Therefore, I also skip a detailed description of the method in this work. The interested reader is referred to [109].

In this section, I give results for QR1 and QR2 for the wall-mounted flap testcase and exemplarily for (GS). The aim is to back-up the results of [109], but also to conclude on robust choices of the filtering including the limit parameter ϵ for changing physical settings.

Algorithm 2 QR2-filtered modified Gram-Schmidt orthogonalization.

```
$R_{kk} = \|V(:,k)\|_2$, $Q(:,k) = V(:,k)/R_{kk}$
for $i = k, k-1, \dots, 2$ do
  $v = V(:,i)$
  for $j = i-1, i-2, \dots, 1$ do
    $R_{ji} = Q(:,j)^T \cdot v$
    $v = v - R_{ji} \cdot Q(:,j)$
  end for
  if $\|v\|_2 < \epsilon \cdot \|V(:,i)\|_2$ then
    Remove column $i$ from $V$ and $W$.
    Start over from top.
  end if
  $R_{ii} = \|v\|_2$, $Q(:,i) = v/R_{ii}$
end for
```

In the FSI community, QR1 is the most widely used technique and was already originally supported in preCICE though with an absolute criterion [99]. In the fixed-point acceleration community, different approaches are used to stabilize the QR-decomposition. Marks and Luke [137] use a regularization term to stabilize the least-squares problem

$$\min_{\alpha \in \mathbb{R}^k} \|V_k \cdot \alpha + R^k\|_2^2 + \frac{\beta}{2}\|\alpha\|_2^2 \ , \ \ \beta > 0 \ .$$

Fang and Saad [78] use a similar approach to POD, but without deleting columns definitely. They, furthermore, restart the iteration[35], i.e., delete all columns, if

$$R^k > \gamma^{-1} R^{k-1}, \ \ \gamma \in [0.1, 0.3] \ .$$

[34] Rob Haelterman, Alfred Bogaers, Miriam Mehl, Klaudius Scheufele and myself.

[35] In [78], the restart formula actually reads $R^k > \gamma R^{k-1}$, but due to the description, I am almost sure that this must be a typing error.

Walker and Ni [217] simply delete the oldest columns if the condition of R drops below a certain limit. All these approaches are valid alternatives, but not too promising for transient problems such as partitioned FSI, as they need methods that filter out particular columns while maintaining as much information from previous timesteps as possible. To this end, recent timesteps with many iterations should be reduced to a robust amount while older timesteps should not be discarded completely. For FSI, restart typically also results in a loss of robustness.

The implementation of QR1 differs slightly from the version of [109] in the sense that it does not recompute the complete QR-decomposition after the deletion of a column, but uses an update scheme based on Given's rotations. Section 4.4 gives details on the implementation in preCICE. Table 10 compares this pure update scheme (Upd) with a modified Gram-Schmidt (GS) re-computation once per iteration. The difference between both approaches is visible for a non-filtered matrix V. Here, modified Gram-Schmidt typically results in a more robust QR-decomposition, since recent iterates are emphasized over older iterates, in contrast to the update scheme. The difference between both approaches vanishes for a well-filtered V. In principle, QR2 could also be implemented by means of an update scheme. preCICE, however uses the original approach and can, thus, offer a cheap variant, QR1, and a more involved, but assumably more robust variant, QR2.

Filter / ϵ	1e-2	1e-3	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9	1e-10	1e-11	1e-12	1e-13
Upd-QR1 its	6.67	3.48	3.29	3.38	3.58	3.43	4.07	4.34	4.65	4.26	4.12	3.98
Upd-QR1 dc	5.46	1.87	1.04	0.41	0.2	0.08	0.12	0.1	0.04	0.09	0.01	0.00
GS-QR1 its	6.41	3.50	3.29	3.37	3.45	3.44	3.57	3.76	5.03	4.41	4.19	4.19
GS-QR1 dc	5.20	1.87	1.04	0.40	0.13	0.08	0.08	0.08	0.18	0.13	0.00	0.00

Table 10: Average number of iterations (its) and deleted columns (dc) for the wall-mounted flap testcase, first 100 timesteps. The QR1-filter with an updated QR decomposition (Upd-QR1) is compared against a full modified Gram-Schmidt orthogonalization (GS-QR1) in every iteration for various limits ϵ. Mesh M2 is used and $\rho_F = 0.1$ compared to the baseline parameters.

Table 11 shows the average number of iterations for QR1 and QR2, compared to a non-filtered QR-decomposition over various physical settings of the wall-mounted flap testcase. For the sake of completeness, Table 12 list the average amount of deleted columns for the same experiments. Both filters manage to stabilize cases that lead to divergence without filtering. In contrast to the results of [109], QR1 shows a better behavior than QR2 in terms of efficiency and also in terms of robustness. This may be due to the significantly more challenging stability problems that the scenario under consideration features compared to the standard benchmarks in [109], for which filtering is more a question of efficiency than of robustness. $\epsilon \in [10^{-5}, 10^{-4}]$ appears to be a good choice for QR1 independent from the physical settings, again in contrast to [109] where much smaller values are optimal. Similar tests for S-GB2 show a significantly less sensitive behavior to the filtering, which comes at no surprise as no columns from previous timesteps are reused while information on the Jacobian approximation is kept implicitly. I conclude from this section and the results of [109] that filtering is a crucial ingredient for Anderson acceleration and very demanding testcases. The QR1 filter shows a robust and efficient behavior for such demanding scenarios while QR2 might outperform QR1 for moderate scenarios.

3.6.3 A Preconditioner for the Jacobian System

The Jacobian system (J) suffers from a possibly bad condition as forces and displacement, concatenated in $x^T = (d^T, f^T)$, may live on very different scales. A remedy to this problem is a weighting respectively pre-conditioning[36] of the least-square system

$$\min_{\alpha \in \mathbb{R}_k} \|\Phi^k \cdot V_k \cdot \alpha + \Phi^k \cdot R^k\|_2 \ ,$$

with the preconditioner $\Phi = \text{diag}(\phi_1, \ldots, \phi_n)$. In the QR-setting, this corresponds to a preconditioning of $V_k \leftarrow \Phi^k V_k$ and $R^k \leftarrow \Phi^k R^k$ for Anderson acceleration. For generalized Broyden, this becomes more involved as also $W_k \leftarrow \Phi^k W_k$ needs to be scaled, which requires a backward scaling of the

[36] I prefer the term preconditioner over the term scaling to not let the reader confuse *numerical* scaling with *HPC* scaling, though the approach presented is in fact a simple scaling of vectors.

	QR1-Filter				QR2-Filter				No Filter	
Testcase / ϵ	1e-3	1e-4	1e-5	1e-6	0.25	1e-1	1e-2	1e-3	Upd	GS
M1	6.14	**4.73**	4.74	5.12	8.74	6.73	**5.29**	div	8.57	10.44
M1 $\rho_S = 1.0$	3.91	**3.71**	3.98	4.14	5.10	**5.00**	5.10	6.54	8.35	8.77
M1 $E = 3 \cdot 10^5$	4.95	**4.40**	4.75	4.74	6.52	5.56	**4.78**	5.13	10.84	div
M1 $\delta t = 2 \cdot 10^{-3}$	8.33	5.81	**5.61**	5.82	8.45	6.81	**5.33**	5.99	6.62	7.43
M1 $u_0 = 5.0$	10.08	**6.60**	7.17	8.21	12.13	7.64	**6.44**	7.39	div	div
M1 $\rho_F = 0.1$	3.4	3.29	**3.22**	3.24	4.88	4.29	3.57	**3.42**	3.45	3.41
M2	9.58	8.28	**7.42**	8.00	13.56	18.37	**12.58**	div	div	div
M2 $\rho_S = 1.0$	**3.82**	4.05	5.49	6.18	9.88	8.73	**7.97**	9.25	div	div
M2 $E = 3 \cdot 10^5$	9.58	8.28	**7.42**	8.00	13.56	18.37	**12.58**	div	div	div
M2 $\delta t = 2 \cdot 10^{-3}$	13.72	7.25	**6.05**	7.76	16.08	div	**12.20**	div	div	div
M2 $u_0 = 5.0$	11.92	**7.93**	8.89	div	div	11.84	**10.39**	div	div	div
M2 $\rho_F = 0.1$	3.48	**3.29**	3.38	3.58	4.97	4.39	4.18	**3.67**	3.98	4.19

Table 11: Average iterations per timestep for the Wall-Mounted Flap testcase with S-AA2(5), first 100 timesteps. Physical settings are only indicated when they are different to the baseline setting. Upd denotes the update QR-decomposition, GS the modified Gram-Schmidt scheme. Best values use a bold type setting.

	QR1-Filter				QR2-Filter				No Filter	
Testcase / ϵ	1e-3	1e-4	1e-5	1e-6	0.25	1e-1	1e-2	1e-3	Upd	GS
M1	4.53	2.16	1.34	0.92	7.58	5.38	3.08	div	0.00	0.00
M1 $\rho_S = 1.0$	1.56	0.51	0.28	0.19	3.47	2.86	2.41	3.46	0.00	0.00
M1 $E = 3 \cdot 10^5$	3.72	2.76	2.40	1.61	5.47	4.49	3.6	3.55	0.00	div
M1 $\delta t = 2 \cdot 10^{-3}$	7.07	3.86	2.73	2.02	7.39	5.67	3.73	3.81	0.00	0.00
M1 $u_0 = 5.0$	8.11	3.55	2.92	2.79	10.99	6.33	4.45	4.24	div	div
M1 $\rho_F = 0.1$	1.80	1.08	0.46	0.05	3.58	2.71	1.43	0.89	0.00	0.00
M2	8.4	6.86	5.33	5.07	12.49	17.32	11.44	div	div	div
M2 $\rho_S = 1.0$	2.07	1.68	2.68	2.6	8.79	7.59	6.58	7.54	div	div
M2 $E = 3 \cdot 10^5$	8.4	6.86	5.33	5.07	12.49	17.32	11.44	div	div	div
M2 $\delta t = 2 \cdot 10^{-3}$	12.41	5.25	3.18	3.8	14.98	div	10.7	div	div	div
M2 $u_0 = 5.0$	9.99	4.87	4.6	div	div	10.41	8.37	div	div	div
M2 $\rho_F = 0.1$	1.87	1.04	0.41	0.2	3.65	2.8	2.0	1.28	0.00	0.00

Table 12: Average amount of deleted columns per timestep for the Wall-Mounted Flap testcase with S-AA2(5), first 100 timesteps. Physical settings are only indicated when they are different to the baseline setting. Upd denotes the update QR-decomposition, GS the modified Gram-Schmidt scheme.

update $\Delta\tilde{x}^k \leftarrow (\Phi^k)^{-1}\Delta\tilde{x}^k$ after each iteration. Furthermore, the previous Jacobian needs to be scaled $J_{\tilde{R}}^{-1,(N)} \leftarrow \Phi^k J_{\tilde{R}}^{-1,(N)}(\Phi^k)^{-1}$. The latter ensures a consistent scaling of both summands in the update formula

$$J_{\tilde{R}}^{-1} = J_{\tilde{R}}^{-1,(N)} + (W_k - J_{\tilde{R}}^{-1,(N)} V_k)(V_k^T V_k)^{-1} V_k^T \; ,$$

introduced in Section 3.2.2. Please note that, if combined with a filter, the preconditioner is applied before the filter.

A straight-forward choice for Φ^k is to normalize all entries. In a transient context this is often done by means of the previous timestep. This can either be done in a per-entry basis

$$\phi_i = (x_i^{(N)} + \epsilon_{abs})^{-1} \; , \; i = 1 \ldots n \; ,$$

where ϵ_{abs} denotes a lower border to avoid division by zero, or in a per-sub-vector basis

$$\phi_i^d = \|d^{(N)}\|_2^{-1} \text{ and } \phi_i^f = \|f^{(N)}\|_2^{-1} \; , \; i = 1 \ldots \frac{n}{2} \; ,$$

where ϕ_i^d and ϕ_i^f denote the weighting factors associated to the displacement and to the force entries, respectively. Both cases necessitate a re-computation of the QR-decomposition from scratch at the beginning of each timestep. Extensive testing shows that the per-sub-vector variant consistently outperforms the per-entry variant. For the wall-mounted flap testcase, in the baseline setting and mesh M1, for example, the first results in 5.67 and the latter one in 6.47 iterations per timestep (over the first 100 timesteps)[37]. I assume that this is due to the destruction of a natural weighting of the degrees of freedom by the per-entry variant. Any vertex has the same influence, ignoring whether it is located on the tip of the flap or close to the lower wall. Therefore, I prefer the per-sub-vector over the per-entry variants. I refer to this first (per sub-vector) weighting as *value* weighting.

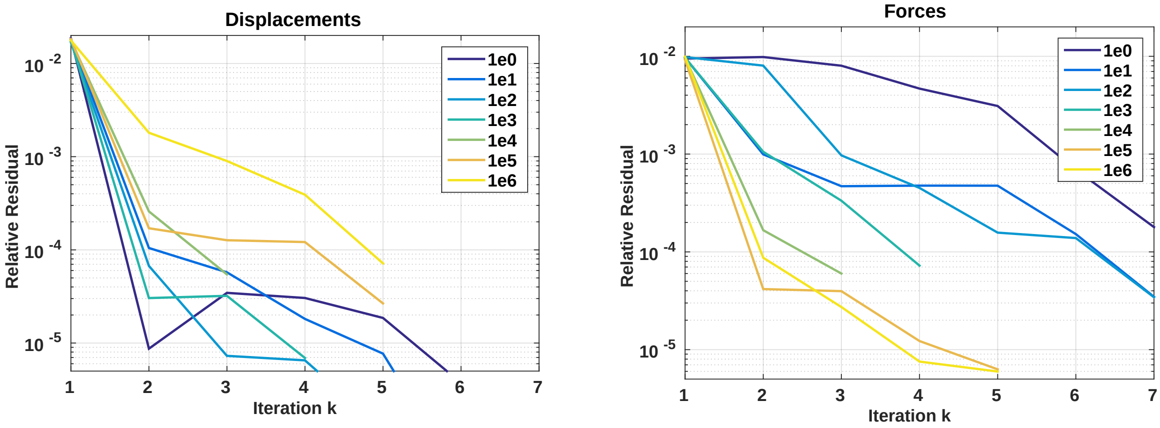


Figure 36: Relative residual for displacements (left) and forces (right) for the wall-mounted flap testcase, at timestep 30, and various constant force weighing factors a.

For a broader comparison, I also introduce a *constant* weighting. This variant weights force entries by a simple constant scaling factor $a \in \mathbb{R}$, $\phi_i^f = 1/a$, $i = 1 \dots n/2$, having the advantage of not requiring any QR recomputations. Figure 36 shows the relative residual of both, displacements and forces, at timestep 30, for the baseline setting and Mesh M1, and for various factors a. The different influence of a on both criteria is clearly visible: the color coding flips comparing both subplots. This means, putting more weight on the forces, i.e. using a smaller a, results in better values for the displacement criterion in the next iteration as $d = S(f)$ and worse values for the force criterion. The best choice of a should, therefore, somehow balance both criteria. This observation leads to the idea of the *residual* weighting

$$\phi_i^d = \|\tilde{d}^k - d^k\|_2^{-1} \text{ and } \phi_i^f = \|\tilde{f}^k - f^k\|_2^{-1}, \; i = 1 \dots \frac{n}{2} .$$

This approach requires a QR-decomposition from scratch in every iteration. For some cases, a full *residual* weighting can overdue the re-weighting, which leads to a zig-zag convergence behavior. To overcome this drawback, I finally introduce the *residual-sum* weighting

$$\phi_i^d = \left(\sum_{j=1}^{k} \frac{\|\tilde{d}^j - d^j\|_2}{\|\tilde{x}^j - x^j\|_2} \right)^{-1} \text{ and } \phi_i^f = \left(\sum_{j=1}^{k} \frac{\|\tilde{f}^j - f^j\|_2}{\|\tilde{x}^j - x^j\|_2} \right)^{-1}, \; i = 1 \dots \frac{n}{2} ,$$

which also requires a QR-decomposition from scratch in every iteration. Marks and Luke also discuss this approach [137], while finally using the square-root of it to further adjust for possible over-scaling. I did not observe such problems for FSI and stick therefore to the original formulation.

Tables 13 and 14 list average iteration numbers for all four approaches, different filter configurations and for J-AA2(5) and J-GB2(2), respectively, over various physical settings of the wall-mounted flap testcase. In general, the QR1 filter shows better results than QR2 for J-AA(5), similar to the results of the last section. For J-GB(2), QR2 tends to outperform QR1 for mesh M1, but shows a drastic loss of efficiency for mesh M2. A better limit parameter ϵ could probably weaken this effect, but this is not the topic of this section. The best value for the constant weighting is relatively stable over the various physical settings and both meshes, only the timestep size seems to have a significant influence. The value weighting is a useful choice as it comes close to the best constant weighting, without reaching it for most cases. Especially for J-GB(2), the performance of the value variant is rather mediocre. The

[37] For sake of readability, I omit complete tables for the per-entry variant.

residual weighting shows promising results for some cases, but tends to be not robust enough. For the QR2 filter, this even leads to divergence for many cases. The residual-sum weighting, finally, seems to be the best choice as it comes close to the best constant variant and even outperforms it regularly. Furthermore, it is worth mentioning, that J-GB(2) shows significant better results than J-AA(5) with the exception of the highest added-mass values and mesh M2.

To conclude this section, I can state that the goal to find a dynamic preconditioner is achieved. Independence from an explicit parameter tuning clearly outweighs the possibly additional cost stemming from, e.g., re-computations of the QR-decomposition. Changing from (GS) to (J) does not result in additional tuning parameters.

		Constant							Dynamic		
Setup / Precond.	Fil.	10^0	10^1	10^2	10^3	10^4	10^5	10^6	Val.	Res.	RS
M1	QR1	8.58	8.28	6.17	5.20	**4.87**	5.30	6.41	5.01	5.54	**4.95**
	QR2	8.88	8.86	7.57	5.37	4.91	5.50	8.41	5.10	5.94	5.01
M1 $\rho_S = 1.0$	QR1	6.79	6.69	5.83	4.71	**4.06**	4.17	4.93	4.38	4.62	4.20
	QR2	6.92	7.07	6.87	5.07	4.10	4.16	4.94	4.43	5.00	**4.13**
M1 $E = 3 \cdot 10^5$	QR1	8.03	6.28	4.84	3.98	**3.71**	3.94	4.69	3.88	4.16	**3.79**
	QR2	8.03	7.91	6.26	4.09	3.76	4.28	7.23	3.97	div	3.87
M1 $\delta t = 2 \cdot 10^{-3}$	QR1	11.75	7.66	5.79	**5.01**	5.04	6.02	7.82	**4.99**	5.36	5.04
	QR2	9.25	9.18	6.21	5.15	5.08	7.28	10.03	5.09	5.68	5.03
M1 $u_0 = 5.0$	QR1	11.04	10.98	8.62	6.73	6.32	7.22	8.96	6.41	7.46	**6.34**
	QR2	11.13	11.12	9.42	6.81	**6.26**	7.16	10.41	6.52	6.68	6.41
M1 $\rho_F = 0.1$	QR1	6.91	4.93	4.12	3.57	3.55	3.96	5.35	3.63	3.84	**3.50**
	QR2	6.71	6.47	4.38	3.63	**3.52**	4.26	5.28	3.74	div	3.59
M2	QR1	9.78	8.57	8.04	6.72	**6.57**	7.91	9.72	**6.70**	7.74	7.22
	QR2	11.95	10.96	6.73	7.99	7.60	8.61	28.17	8.17	8.64	8.55
M2 $\rho_S = 1.0$	QR1	9.46	9.50	7.74	7.22	**6.21**	7.63	8.15	**7.21**	8.39	8.52
	QR2	9.95	9.77	8.51	6.40	8.75	10.14	9.84	9.37	div	9.10
M2 $E = 3 \cdot 10^5$	QR1	8.75	6.53	5.19	4.77	**4.44**	5.19	6.21	4.71	**4.24**	4.85
	QR2	9.12	8.74	6.04	5.07	5.40	7.04	20.25	6.07	5.34	6.60
M2 $\delta t = 2 \cdot 10^{-3}$	QR1	8.84	6.52	5.57	**5.10**	5.67	7.19	div	5.35	6.31	**5.25**
	QR2	9.35	8.39	5.88	5.23	6.30	14.03	div	5.48	div	5.70
M2 $u_0 = 5.0$	QR1	12.38	10.89	8.32	7.40	**7.18**	9.30	11.94	**7.12**	7.47	7.18
	QR2	12.55	12.42	8.76	7.34	7.40	10.32	19.50	7.66	7.80	7.14
M2 $\rho_F = 0.1$	QR1	5.72	4.95	3.89	**3.50**	3.91	4.59	6.51	3.74	3.86	**3.62**
	QR2	6.11	5.48	4.03	3.58	4.00	5.53	6.40	3.97	div	3.69

Table 13: Average number of iterations over the first 100 timesteps for J-AA2(5), the wall-mounted flap testcase, and various preconditioner approaches. Physical settings are only indicated when they are different from the baseline setting. The filters use $\epsilon = 10^{-5}$ for QR1 and $\epsilon = 10^{-2}$ for QR2. Best values are displayed in bold. RS stands for residual-sum.

		Constant							Dynamic		
Setup / Precond.	Fil.	10^0	10^1	10^2	10^3	10^4	10^5	10^6	Val.	Res.	RS
M1	QR1	9.06	9.42	6.01	4.18	4.00	4.61	6.74	5.26	4.54	4.02
	QR2	9.59	9.04	6.45	4.74	**3.92**	4.97	7.05	5.15	div	**3.93**
M1 $\rho_S = 1.0$	QR1	7.18	6.67	5.92	3.81	3.21	3.67	4.64	4.54	4.17	3.47
	QR2	8.18	7.31	6.45	4.71	**3.16**	3.53	4.84	4.56	div	**3.21**
M1 $E = 3 \cdot 10^5$	QR1	8.51	6.76	5.43	3.68	3.55	3.58	4.76	4.68	**3.33**	3.46
	QR2	8.19	8.27	5.28	4.39	**3.34**	3.58	5.36	4.68	div	3.40
M1 $\delta t = 2 \cdot 10^{-3}$	QR1	10.88	8.73	6.25	**4.45**	4.65	5.77	7.78	5.50	4.63	4.59
	QR2	10.38	9.20	5.75	4.67	4.49	5.96	10.56	5.17	4.75	**4.35**
M1 $u_0 = 5.0$	QR1	13.08	div	div	6.21	5.48	6.81	9.05	7.39	8.48	5.93
	QR2	12.27	12.19	8.42	5.98	**5.44**	6.83	9.80	7.36	6.57	**5.59**
M1 $\rho_F = 0.1$	QR1	7.09	4.90	3.49	**2.87**	2.87	3.46	5.51	4.51	4.41	2.87
	QR2	6.38	5.55	3.75	2.88	2.89	4.04	4.99	4.37	div	**2.85**
M2	QR1	10.65	9.80	8.85	**8.54**	9.12	10.29	12.79	9.66	11.42	**8.91**
	QR2	22.36	11.86	13.28	15.51	14.78	12.37	23.6	div	19.46	13.85
M2 $\rho_S = 1.0$	QR1	11.04	11.43	**8.98**	9.10	9.03	12.09	13.86	11.06	11.53	**10.64**
	QR2	12.59	13.27	11.49	11.16	18.26	13.12	div	div	div	15.17
M2 $E = 3 \cdot 10^5$	QR1	9.23	8.51	6.30	7.15	6.09	**5.25**	8.43	8.40	**6.13**	6.47
	QR2	12.53	10.26	9.52	10.09	7.97	8.81	15.21	div	11.20	14.05
M2 $\delta t = 2 \cdot 10^{-3}$	QR1	9.05	6.47	6.56	5.05	5.20	7.05	div	6.21	5.55	**4.70**
	QR2	10.56	7.58	7.93	**4.55**	5.68	10.34	div	6.24	div	5.83
M2 $u_0 = 5.0$	QR1	16.57	11.19	8.18	7.30	**6.54**	10.24	12.91	7.73	6.83	**6.25**
	QR2	15.02	12.89	9.57	13.69	8.25	9.41	22.57	7.68	14.34	9.52
M2 $\rho_F = 0.1$	QR1	5.49	4.72	3.34	**2.84**	3.21	4.92	6.77	4.40	5.01	3.12
	QR2	6.64	4.66	3.24	2.88	3.61	4.97	7.08	4.43	div	**2.90**

Table 14: Average number of iterations over the first 100 timesteps for J-GB2(2), the wall-mounted flap testcase, and various preconditioner approaches. Physical settings are only indicated when they are different from the baseline setting. The filters use $\epsilon = 10^{-5}$ for QR1 and $\epsilon = 10^{-2}$ for QR2. Best values are displayed in bold. RS stands for residual-sum.

3.7 Advanced Numerical Experiments

After some specific preliminary results in the last section, this section gives complete tests for the coupling schemes for established benchmark scenarios. I revisit and test the robustness of the filters, introduced in Section 3.6.2, and the preconditioners, introduced in Section 3.6.3. Also, I study the influence of the number of explicitly reused timesteps. Most important, I conclude on the competitive position of parallel coupling schemes compared to their classical serial counterparts.

I choose three testcases to encompass a broad spectrum of different physical FSI behaviors, such as an enclosed structure, an outer structure and also a membrane. Similarly, I cover a significant spectrum of available single physics solvers, including various numerical schemes. These three testcases are introduced in Sections 3.7.1 to 3.7.3, while Section 3.7.4 collects all results and draws final conclusions. Table 15 overviews the three scenarios. Please compare Section 2.4 for details of all single physics solvers and their numerical schemes. Table 16 gives an overview on the material and numerical parameters of all scenarios.

Abbreviation	Scenario Description	Section	Fluid Solver	Structure Solver
FSI3	Elastic cantilever in 2D channel flow	3.7.1	Alya Nastin	Alya Solidz
3D-Tube	Travelling pressure wave in elastic 3D tube	3.7.2	OpenFOAM	OpenFOAM
DC	3D lid-driven cavity with flexible bottom	3.7.3	Alya Nastin	Carat++

Table 15: Overview on the advanced numerical test scenarios and their solvers.

To judge on the convergence speed of various coupling schemes, I use a relative convergence criterion for both, forces and displacements, of 10^{-4}, compare the discussion of Section 3.1. All single physics solvers use sufficiently tight tolerances for their inner iterations, at least a relative criterion of 10^{-6}, so two orders of magnitude tighter. The quasi Newton schemes use a maximum of 100 columns in their matrices V and W. Oldest columns are dropped, in case a scheme exceeds this number. As initial guess of every timestep, I simply reuse the converged iterate of the previous timestep, which relates to a 0-order extrapolation. If a coupling scheme exceeds the maximum of 50 iterations per timestep, I regard the test as divergent. The very first iteration uses an underrelaxation with parameter 0.5. All physical results show no visible difference if two coupling schemes are compared to each other. [142] gives further physical validation.

		FSI3	3D-Tube	DC
fluid density	ρ_F	$1.0 \times 10^3\,\mathrm{kg/m^3}$	$1.0 \times 10^3\,\mathrm{kg/m^3}$	$1.0\,\mathrm{kg/m^3}$
dynamic viscosity	μ	$1.0\,\mathrm{kg/(m\,s)}$	$3.0 \times 10^{-3}\,\mathrm{kg/(m\,s)}$	$3.0 \times 10^{-3}\,\mathrm{kg/(m\,s)}$
reference velocity	u_0	$2.0\,\mathrm{m/s}$	-	$1.0\,\mathrm{m/s}$
structure density	ρ_S	$1.0 \times 10^3\,\mathrm{kg/m^3}$	$1.2 \times 10^3\,\mathrm{kg/m^3}$	$5.0 \times 10^2\,\mathrm{kg/m^3}$
Young's modulus	E	$5.6 \times 10^6\,\mathrm{N/m^2}$	$3.0 \times 10^5\,\mathrm{N/m^2}$	$250\,\mathrm{N/m^2}$
Poisson ratio	ν	0.4	0.3	0
timestep size	Δt	$1.0 \times 10^{-3}\,\mathrm{s}$	$1.0 \times 10^{-4}\,\mathrm{s}$	$1.0 \times 10^{-2}\,\mathrm{s}$

Table 16: Advanced numerical test scenarios: physical parameters.

3.7.1 FSI3 Benchmark

The FSI3 benchmark was proposed by Turek and Hron [199] following a common effort of the *DFG Forschergruppe* 493[38]. The benchmark consists of a 2D incompressible channel flow with a fixed, rigid cylinder and an attached elastic cantilever placed inside. The cylinder position is slightly off-centric to foster oscillations. Figure 37 sketches the geometry.

[38] http://fsw.informatik.tu-muenchen.de

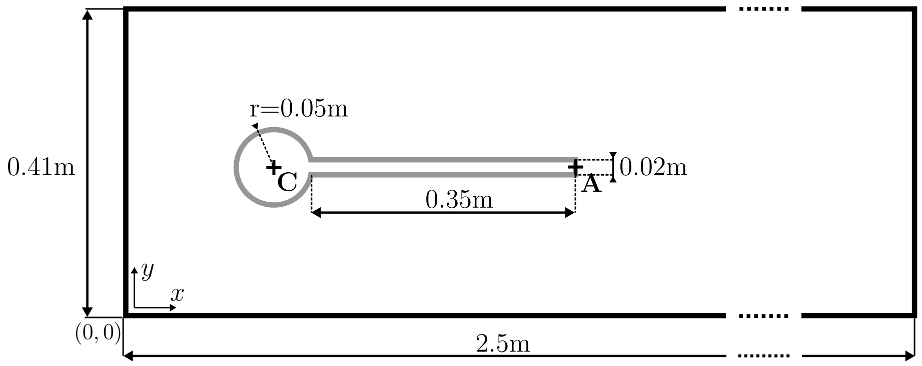


Figure 37: Geometrical layout of the FSI3 benchmark scenario. Point $C = (0.2\text{m},0.2\text{m})$ marks the position of the cylinder's center, which is slightly off-centric in y direction to foster oscillations. Measurements of displacements are done at the center of the backside of the cantilever, point $A = (0.6\text{m},0.2\text{m})$. On the left boundary a parabolic inflow profile is prescribed, the right boundary marks an outflow. No-slip walls are used for the top and bottom boundary as well as on the geometry. The picture is taken from [99].

At the inflow, a parabolic velocity profile in x direction with mean u_0 is prescribed. Originally, [199] describes three variants of this benchmark, which differ in their used physical parameters. I solely use the third variant as it marks the strongest added-mass effect. Table 16 lists the physical parameters. This third variant leads to an unstationary flow with regular oscillations of the cantilever, which can then be compared to other numerical reference simulations. As initial values, I use a precomputed fluid field. This renders an up-ramping of u_0 unnecessary.

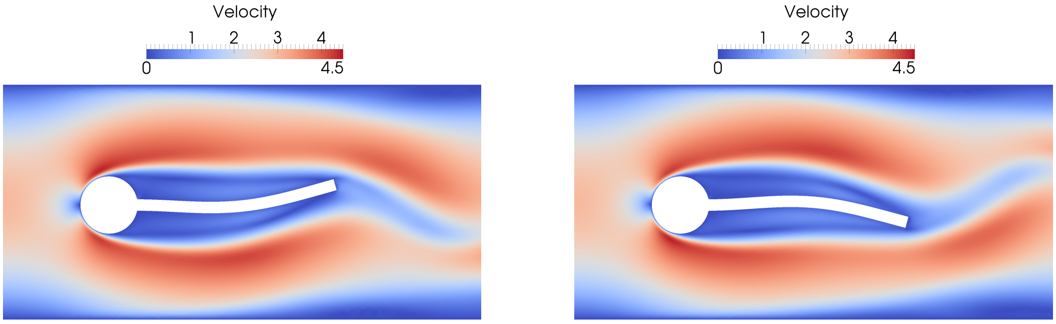


Figure 38: FSI3 testcase: velocity magnitude and structure deformation for $t = 4.13\,\text{s}$ (left) and $t = 4.23\,\text{s}$ (right).

I use two different meshes for the FSI3 testcase - a fine and a coarse configuration. For the fine configuration, the fluid and structure mesh consist of 69460 and 15850 triangular elements, respectively. For the coarse configuration, there are 7928 and 702. For both configurations, the fluid and the structure mesh match at the coupling interface. Figure 38 shows the flow and the deformed structure at two instances during the oscillation for the fine configuration. Figure 39 shows the corresponding displacement at the reference point A during several oscillations and Table 17 compares the frequency and magnitude of this oscillations to reference values. For the displacement at the reference point, good agreement is achieved. [199] also proposes reference values for the dimensionless force acting on the complete geometry. With Alya Nastin, I could not reproduce these force values. I suspect that this is a simple post-processing issue with no physical influence, considering the correct displacement values. `preCICE` as well as the applied coupling schemes can be excluded as error source since a Fluent-preCICE-COMSOL coupling showed good agreement also in the forces values [142]. To conclude upon the various applied coupling schemes for the FSI3 testcase, I consider average iteration numbers per timestep between $t = 2.0\,\text{s}$ and $t = 3.0\,\text{s}$.

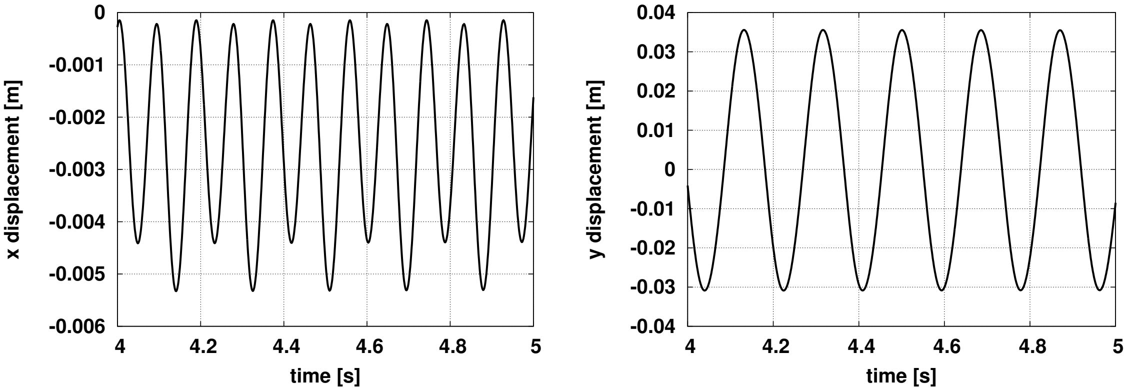


Figure 39: FSI3 testcase, fine mesh configuration: x and y position of the cantilever at point A between $t = 4.0\,\mathrm{s}$ and $t = 5.0\,\mathrm{s}$.

	x displacement $[10^{-3}m]$	y displacement $[10^{-3}m]$
Nastin-Solidz	-2.53 ± 2.34 [10.8]	2.34 ± 33.23 [5.4]
Fluent-COMSOL [142]	-2.50 ± 2.29 [10.9]	1.71 ± 31.94 [5.5]
Reference [199]	-2.69 ± 2.53 [10.9]	1.48 ± 34.38 [5.3]

Table 17: FSI3 testcase, fine mesh configuration: displacement values at point A. The values are given as mean ± amplitude [frequency] and are computed from the first full oscillation after $t = 4s$. Comparison to a `Fluent-COMSOL` run as well as the original reference shows good agreement.

3.7.2 Straight 3D Elastic Tube

As a second testcase, I use the wave propagation in a straight, 3D, elastic tube. The benchmark scenario is described, e.g., in [91], but appeared, to my best knowledge, first in [93]. The scenario setup is provided as part of the preCICE OpenFOAM adapter[39] by David Blom. Figure 40 sketches the geometry of the scenario. Both ends of the tube are fixed.

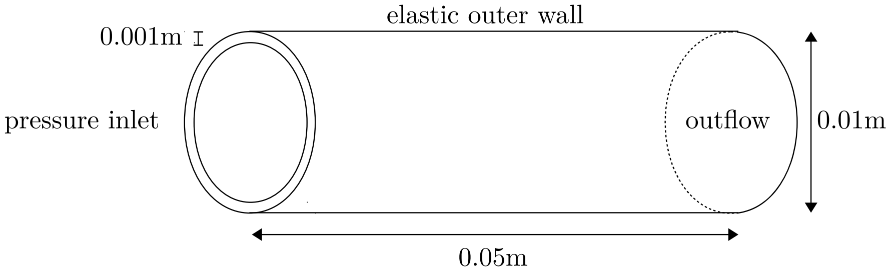


Figure 40: 3D Tube: geometry. Both ends of the tube are fixed. An initial pressure wave is prescribed at the inlet and travels through the domain.

Till $t = 0.003\,\mathrm{s}$, the pressure boundary condition at the inlet is set to a fixed values of $1.3 \times 10^3\,\mathrm{N/m^2}$. Afterwards, it is set to zero. At the outlet, the pressure is always set to zero. This leads to a pressure wave propagating through the tube. Table 16 again lists the physical parameters. Due to symmetry, the simulation is restricted to a quarter of the tube. The fluid mesh and solid mesh consist of 16000 and 800 hexahedral cells, respectively. At the interface, both meshes match. The complete simulation covers the wave propagating once through the domain during $1 \times 10^{-2}\,\mathrm{s}$. The mean iteration numbers are averaged over the complete 100 timesteps. Figure 41 shows the physical results during four instances.

[39] `https://github.com/davidsblom/FOAM-FSI`

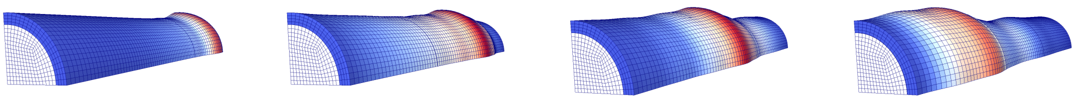


Figure 41: 3D Tube: structure deformation and block-structured mesh, from left to right at $t = 2.5 \times 10^{-3}\,\mathrm{s},\ 5.0 \times 10^{-3}\,\mathrm{s},\ 7.5 \times 10^{-3}\,\mathrm{s},\ 1.0 \times 10^{-2}\,\mathrm{s}$. For sake of visibility, the structure deformation is scaled by a factor of 10.

3.7.3 Driven Cavity with Flexible Bottom

The third and last test scenario originates from a classical CFD benchmark: a lid-driven cavity – incompressible flow over a cavity is modeled as a moving wall. [218] generalized this test scenario to an FSI case by making the bottom wall elastic and the lid velocity oscillating. To not violate the incompressibility, small gaps for inflow and outflow were added. I use the exact same setup as in [218], but generalize the case to 3D. For the boundary in the front and in the back, I use also no-slip walls. Figure 42 depicts the geometry. Again, Table 16 lists the physical parameters. The oscillating Dirichlet condition for the x velocity at the lid reads

$$u_x = u_0 \cdot \left(1 - \cos\left(2\pi \frac{t}{T}\right)\right) \ ,$$

while the y and z components are set to 0. One full period is set to $T = 5.0\,\mathrm{s}$. The inflow condition is a linear interpolation from the full lid velocity at $y = 1.0\,\mathrm{m}$ to a no-slip condition at $y = 0.9\,\mathrm{m}$. Initially, all quantities are set to zero. This is physically meaningful as the lid velocity also starts from 0.

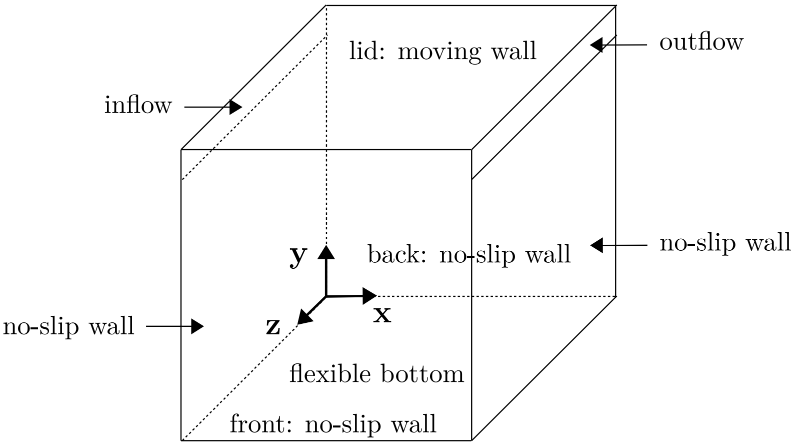


Figure 42: Driven cavity with flexible bottom: geometry and boundary conditions. The cube's side length is 1.0m, while the height of the inflow and outflow measures a tenth of the side length.

The fluid solver uses a equidistant hexahedral mesh with $20 \times 20 \times 20$ elements. The membrane solver uses a matching mesh, thus, 20×20 elements. Figure 44 shows various screenshots during one complete oscillation of the lid velocity. To get a better impression on the initial phase of the simulation, Figure 43 visualizes the vertical displacement of the center point of the membrane. The mean iteration numbers are average of the first 1000 timesteps, i.e. $[0, 2T]$.

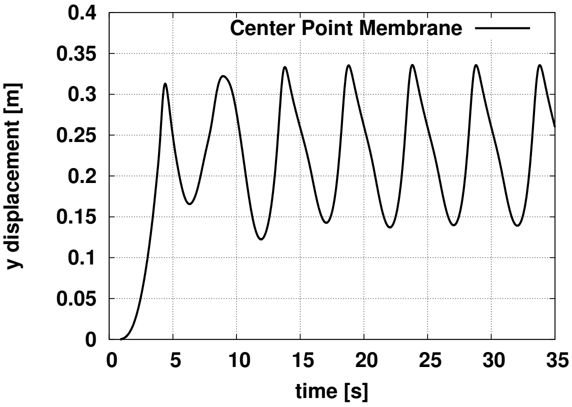


Figure 43: Driven cavity with flexible bottom: vertical displacement of the center point of the membrane $(0.5, 0.0, 0.5)$ over time.

3.7.4 Results and Conclusions

I explain the numerical results in several test series. First, I revisit the filtering concept of Section 3.6.2. To this end, GS-AA5 and GS-GB0, are compared for various filters and all three scenarios. Table 18 lists the mean iteration numbers. Several conclusions can be drawn. First and most important, filtering is indeed a crucial ingredient of a quasi-Newton scheme. Compared to the study in [109], an even higher influence is visible here. Comparing the FSI3 results of [109] with the ones of this work, it becomes clear that even more than the scenario, the applied solvers influence the best filter choice and its impact. [109] uses `OpenFOAM` as the fluid solver, which uses a quite sophisticated mesh movement technique based on radial basis functions. On the other hand, this work uses Alya Nastin as the fluid solver, which uses a simple Laplace smoothing for the mesh movement. Besides many other minor differences, this is probably the most important one. The results of Table 18 show that the Alya Nastin - Alya Solidz coupling benefits highly from the filtering. With too little filtering, the FSI3 testcases diverges for both mesh configurations. The right amount of filtering, however, stabilizes the simple mesh movement. Too much filtering, on the other hand, also leads to divergence, compare FSI3-fine with the QR1 $\epsilon = 10^{-4}$ filter. For the other two testcases, filtering has not a crucial impact on stability, but definitely on the efficiency. For DC and GS-AA, for example, a well-fitted filtering can improve the mean iteration number from 5.50 to 3.01. Which filter works best depends on the testcase. For the 3D-Tube testcase, the QR2 filter shows a better performance, while QR1 outperforms QR2 for the other two scenarios. Furthermore, also for GS-AA5 and GS-GB0, a different behavior can be observed. For example, for the DC testcase, GS-GB0 performs best without any filter, whereas GS-AA5, as mentioned above, shows a clear performance boost from a well-adjusted filter. I conclude that, in general, a medium-tight filter is a good choice. Therefore, I use QR1 with $\epsilon = 10^{-6}$ for all further experiments in this section, except the GB for FSI3-fine, where QR1 with $\epsilon = 10^{-5}$ appears to be the more robust choice. However, for further testcases and especially for further coupled solvers, a brief study upon the best filter configuration is always worth considering.

Next, I study the influence of explicitly reused timesteps on the performance of AA and GB. Table 19 lists mean iteration numbers for AA and GB, for both, (GS) and (J), and for all three testcases. Again, several interesting aspects are visible. AA clearly needs a good tuning of the number of reused timesteps R. Here, (J) can benefit from a higher R than (GS). For the latter, a too high R has a slight negative influence. This negative influence is, however, not as drastic as in preliminary studies, compare Section 3.4 and [176, 142], as apparently the filter can lower such problems. In general, for GS-AA and J-AA a value of 10 and 20, respectively, tends to be a good and robust choice.

Now, let us look at the influence of R on the performance of the GB scheme. First, I want to remind the reader about the concepts of Section 3.2: the motivation behind GB is to implicitly reuse past information and therefore render an explicit tuning of R unnecessary. Indeed, GB0 clearly outperforms AA0. However, for FSI3-fine and DC, explicitly reused past information has still a beneficial influence, but in a clear lower extent than for AA. For the 3D-Tube already $R = 1$ reduces the efficiency. In principle, the standard and conservative choice for GB should be $R = 0$. In this case, but probably only

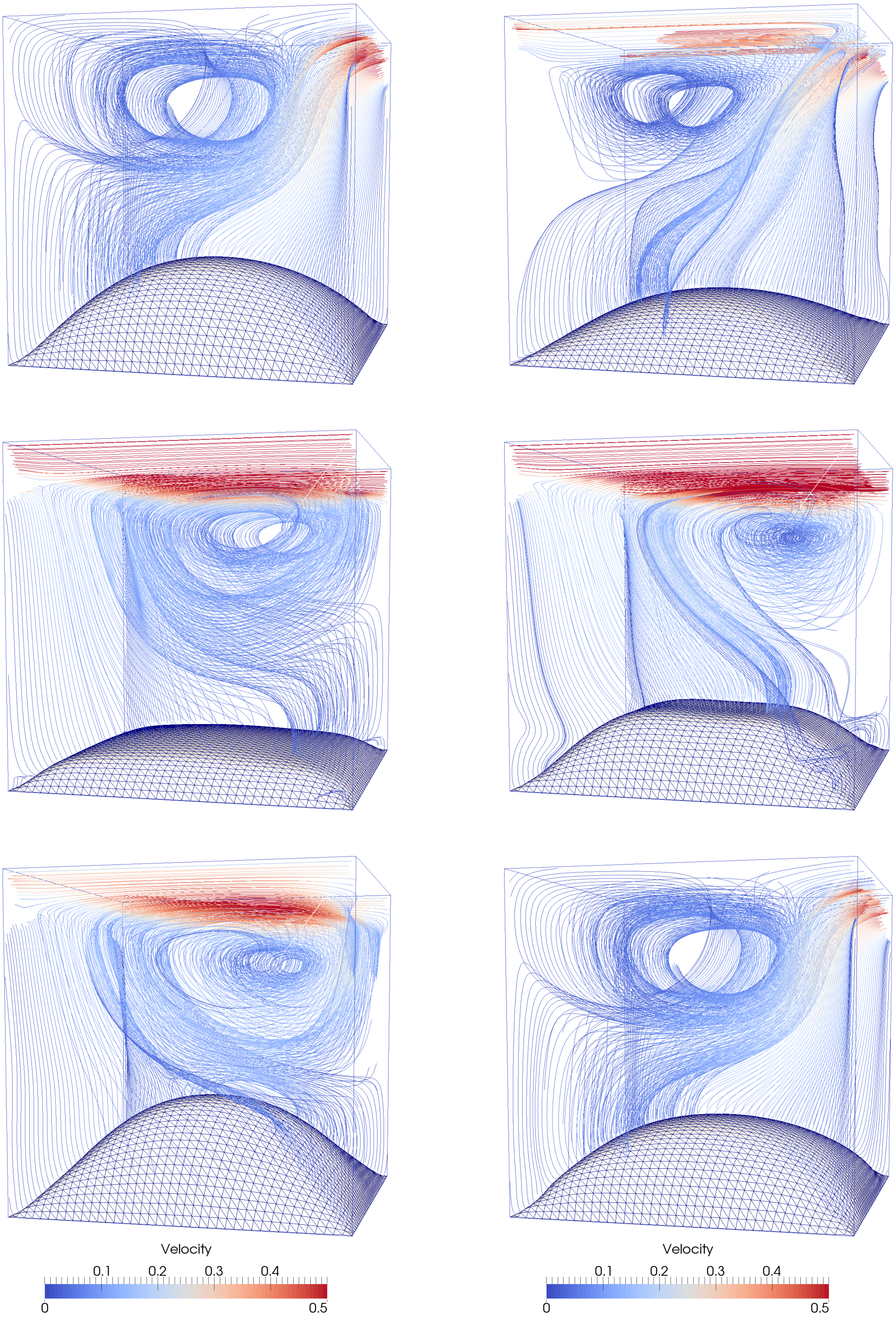


Figure 44: Driven cavity with flexible bottom: membrane deformation and streamlines during one flow period, from left to right and from top to bottom at $t = 10\,\mathrm{s}$, $11\,\mathrm{s}$, $12\,\mathrm{s}$, $13\,\mathrm{s}$, $14\,\mathrm{s}$, $15\,\mathrm{s}$.

then, GB is an important alternative to AA.

I also briefly revisit the preconditioner for the parallel (J) system. Table 20 lists the mean iteration numbers for J-AA20 and J-GB0 for the *residual-sum* and the *value* weighting for all three testcases. The *residual-sum* weighting almost constantly outperforms the *value* weighting, not tremendously, but

		QR1-Filter					QR2-Filter		
Scenario	Method / ϵ	1e-4	1e-5	1e-6	1e-7	1e-15	1e-1	1e-2	1e-3
FSI3-fine	GS-AA(5)	*div*	4.52	4.01	*div*	*div*	*div*	*div*	*div*
	GS-GB(0)	5.91	7.12	*div*	*div*	*div*	6.17	*div*	*div*
FSI3-coarse	GS-AA(5)	11.65	4.86	4.10	4.15	*div*	7.60	8.48	8.86
	GS-GB(0)	5.76	5.75	5.89	5.89	5.89	5.90	5.87	5.92
3D-Tube	GS-AA(5)	15.21	8.78	7.64	7.79	7.87	6.96	8.26	7.71
	GS-GB(0)	8.48	8.27	7.88	7.74	7.74	8.00	7.86	7.75
DC	GS-AA(5)	3.15	3.01	3.02	3.15	5.50	4.42	4.02	4.05
	GS-GB(0)	3.88	4.64	4.46	3.84	3.85	4.36	4.95	3.83

Table 18: Average number of iterations per timestep for various filter configurations and all advanced test scenarios. *div* marks divergent configurations.

Scenario	Method / R	0	2	5	10	15	20	30*	40*
FSI3-fine	GS-AA(R)	10.03	5.31	4.01	3.71	3.81	3.96		
	GS-GB(R)	7.12	5.33	5.37	5.31	6.06	9.77		
	J-AA(R)	14.87	6.48	4.48	3.63	3.38	3.30	3.33	3.49
	J-GB(R)	5.54	4.37	4.00	3.74	3.94	3.94		
FSI3-coarse	GS-AA(R)	10.67	5.50	4.09	3.91	4.05	4.36		
	GS-GB(R)	5.89	6.86	6.08	7.30	6.61	6.36		
	J-AA(R)	16.72	6.45	4.62	3.98	3.78	3.73	3.91	4.18
	J-GB(R)	5.37	3.81	3.58	3.85	4.27	4.67		
3D-Tube	GS-AA(R)	14.40	9.72	7.64	7.45	8.12	8.39		
	GS-GB(R)	7.88	10.19	10.49	12.12	12.34	12.42		
	J-AA(R)	26.16	15.79	12.50	10.80	10.46	10.36	14.14	16.26
	J-GB(R)	9.61	23.09	20.26	19.42	19.42	19.42		
DC	GS-AA(R)	4.64	3.62	3.02	3.00	3.01	3.02		
	GS-GB(R)	4.46	4.03	3.08	3.04	3.02	3.03		
	J-AA(R)	5.47	3.13	2.49	2.18	2.07	2.04	2.01	2.01
	J-GB(R)	3.34	2.34	2.06	2.01	2.01	2.01		

Table 19: Average number of iterations per timestep for different number of reused timesteps R and all advanced test scenarios. All configurations use a QR1 filter with $\epsilon = 10^{-6}$, except the GB schemes for FSI3-fine where a QR1 filter with $\epsilon = 10^{-5}$ is used. All (J) schemes use the *residual-sum* weighting as preconditioner. *: for J-AA addtional runs with an even higher R are included to illustrate the negative influence of a too high R. For these runs, the maximum amount of columns in V and W is increased to 500.

significantly. Only for J-AA20 and the 3D-Tube, the latter shows a slightly better performance.

	J-AA(20)		J-GB(0)	
Scenario / Precond.	res.-sum	value	res.-sum	value
FSI3-fine	3.30	3.84	5.54	8.63
FSI3-coarse	3.73	4.43	5.37	5.28
3D-Tube	10.36	9.90	9.61	10.93
Cavity	2.04	2.49	3.34	4.70

Table 20: Average number of iterations per timestep, comparison between the *residual-sum* and the *value* weighting. All configurations use a QR1 filter with $\epsilon = 10^{-6}$, except the GB scheme for FSI3-fine where a QR1 filter with $\epsilon = 10^{-5}$ is used.

Finally, I want to conclude on the most important contribution of this chapter, the parallel coupling schemes and their competitive position compared to the classical serial counterparts. Table 21 compares the mean iteration numbers of (GS) and (J) for the best value of R for each scenario. The parallel coupling schemes not only match the performance of the serial coupling schemes, but even outperform them for the FSI3 and the DC testcase. Only for the 3D-Tube, the parallel schemes show a slight worse performance. Please remember that a slight worse performance in terms of iterations does not induce a worse overall performance, as the parallel schemes allow for a simultaneous execution of the fluid and the structure solver. Section 5.2 gives a concrete runtime comparison.

Furthermore, I conclude from Table 21 that AA and GB show quite similar performance, in general. Only for the FSI3 testcase, AA significantly outperforms GB. For the fine mesh configuration, this becomes even more noticeable. The OpenFOAM runs of [176] show, however, a different behavior. Therefore, Table 21 also lists OpenFOAM runs with an identical coupling configuration. Last, Table 21 also lists mean iteration numbers for an Aitken underrelexation. The latter is drastically outperformed by the quasi-Newton methods, for the FSI3 testcase by approximately a factor of 4, for the DC testcase by a factor of 2.5. The 3D-Tube even diverges for an Aitken underrelaxation. I included these results specifically to raise awareness when reading publications that compare a monolithic FSI approach to a partitioned Aitken-based one such as, e.g., [180].

		Serial Coupling (GS)			Parallel Coupling (J)	
Scenario	#IV	AA(10)	GB(0)	Aitken	AA(20)	GB(0)
FSI3-fine	719	3.71	7.12	17.00*	3.30	5.54
FSI3-coarse	145	3.91	5.89	18.22	3.73	5.37
FSI3 OpenFOAM	168	7.75	5.84	-	7.70	6.61
3D-Tube	1600	7.45	7.88	div	10.36	9.61
Cavity	400	3.00	4.46	7.37	2.04	3.34

Table 21: Average number of iterations per timestep, comparison between serial and parallel coupling schemes. For each method, the best values of reused timesteps is used and displayed in brackets. All configurations use a QR1 filter with $\epsilon = 10^{-6}$, except the GB schemes for FSI3-fine where a QR1 filter with $\epsilon = 10^{-5}$ is used. For further comparison, also a OpenFOAM FSI3 run is included [176]. #IV is the number of interface vertices. *: only averaged over timesteps 2000 to 2170.

3.8 Generalization to Multi-Coupling

The idea that leads from the classical (GS) to the novel (J) based coupling schemes, can be further applied to generalize the (J) schemes to multi-coupling schemes. By multi-coupling schemes, I mean algorithms that allow for a coupling of more than two single-physics solvers, in contrast to bi-coupling schemes, which only couple two single-physics solvers. The idea that drives the parallel (J) coupling schemes, is the concatenation of different sub-vectors to one global coupling vector. For (J), these are force and displacement sub-vectors. Now, to get to multi-coupling scheme, I concatenate multiple force and displacement sub-vectors from various solvers to get to a global fix-point equation. Solving this fix-point equation with the same techniques that I use for bi-coupling schemes, leads to a robust overall coupling, which furthermore allows for a parallel execution of all involved solvers. In principle, this a simple idea. A general notation, however, is tedious and tends to make this topic more complicated than it actually is. Therefore, I use a simple three-field example in this section to introduce these multi-coupling schemes.

This example consists of two channel flows in opposed directions, separated by an elastic wall. I refer to this fluid-structure-fluid testcase as FSF and explain details in Section 3.8.1. Afterwards, Section 3.8.2 discusses various solution variants including the aforementioned generalized (J) coupling schemes. Finally, Section 3.8.3 presents numerical results and draws conclusions. Besides the FSF testcase, I then also use a second testcase, consisting of a channel flow with four immersed structures. I refer to this second testcase as F4S. The multi-coupling algorithms were first discussed in [204], including results for the FSF testcase. Furthermore, [44] presents results for both testcases. Both publications, however, do not consider the newly developed automatic preconditioner concepts as well as generalized Broyden schemes.

3.8.1 Fluid-Structure-Fluid Model Problem

Figure 45 depicts the geometry of the FSF model problem: two opposed channel flows, $F1$ and $F2$, are separated by an elastic wall, S. To study various intensities of interactions, I vary the width of the wall δ as well as the densities of all three fields $\rho_{F1}, \rho_S, \rho_{F2}$. At both inflow boundaries, an oscillating parabolic velocity profile, which points in the respective flow direction, is prescribed. The oscillation reads $1.0 - \cos(2\pi \cdot t/T)$ with the period $T = 0.1\,\text{s}$ and a timestep size of $\Delta t = 1.0 \times 10^{-3}\,\text{s}$. The total simulation time is $[0, 5T]$.

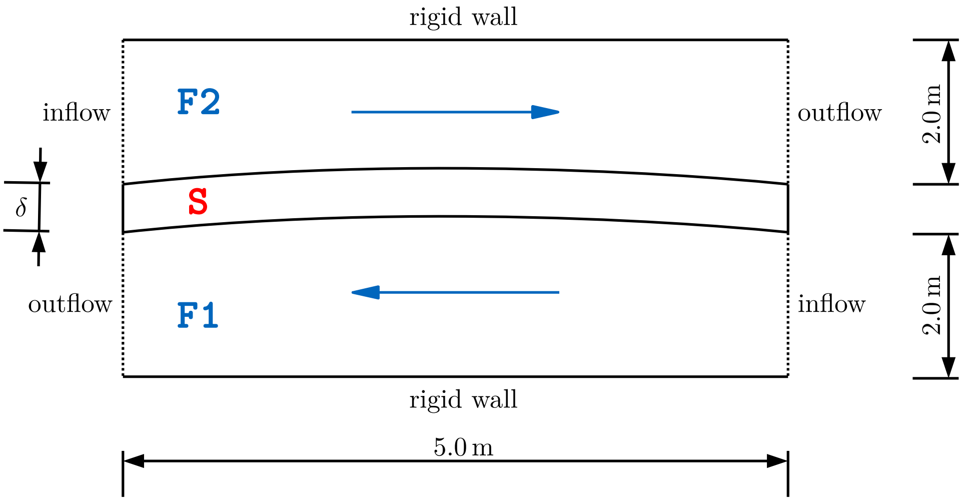


Figure 45: Fluid-structure-fluid model problem: geometry sketch.

The dependencies between all involved physical fields, can be modeled as a directed graph, see Figure 46. Here, the vertices correspond to the physical fields, whereas the edges model the dependencies stemming from the coupling variables. The coupling variables are the force and displacement vectors at the coupling interfaces, f_1, f_2, d_1, d_2. In the following section, this dependency graph is used to visualize various multi-coupling strategies.

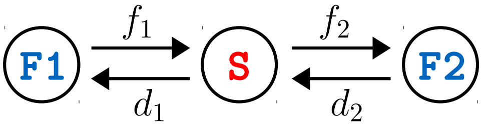


Figure 46: Fluid-structure-fluid model problem: dependency graph.

3.8.2 Various Solution Attempts

Composition of Bi-Coupling Schemes The simplest idea to formulate a coupling strategy for the FSF model problem is to use a bi-coupling scheme for each fluid-structure interaction separately. Figure 47 shows this concept.

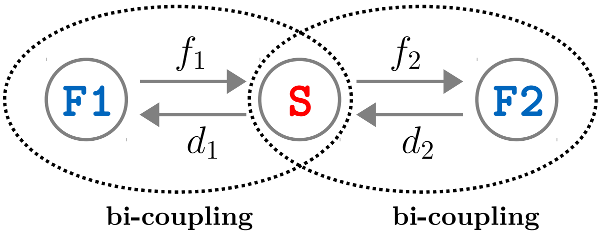


Figure 47: Fluid-structure-fluid model problem: concatenation of bi-coupling schemes.

Each bi-coupling scheme can be adjusted to specific needs of the corresponding interaction. Whether we choose an implicit or an explicit, a serial or a parallel bi-coupling scheme, this results in a different overall execution order. Figure 48 collects several variants. I refer to such schemes by the notation GS-AA5 / GS-EX, if, for example, an implicit GS-AA5 scheme is combined with a serial explicit scheme. Such schemes are implemented in preCICE via a composition pattern and already documented in [99], compare also Figure 32.

If two implicit schemes are combined, compare the lower row of Figure 48, both schemes do not necessarily converge at the same iteration. In this case, the converged scheme simply idles till the second scheme converges as well. Solving both implicit schemes by a quasi-Newton solver, however, results in an inconsistency. To grasp this inconsistency, consider two serial-implicit schemes, as in Figure 48 lower row, left scheme. We aim to solve the fixed-point equation

$$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = S \circ \begin{pmatrix} F1 & 0 \\ 0 & F2 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} ,$$

by decomposing it into two fixed-point equations

$$\begin{aligned} d_1 &= S_{f_2} \circ F1(d_1) \text{ and} \\ d_2 &= S_{f_1} \circ F2(d_2) \, . \end{aligned}$$

I use the indices f_1 and f_2 to point out the dependency of the structure operator S on these values. Now, both fixed-point equations are solved separately, by constructing an approximate Jacobian for $F1$-S as

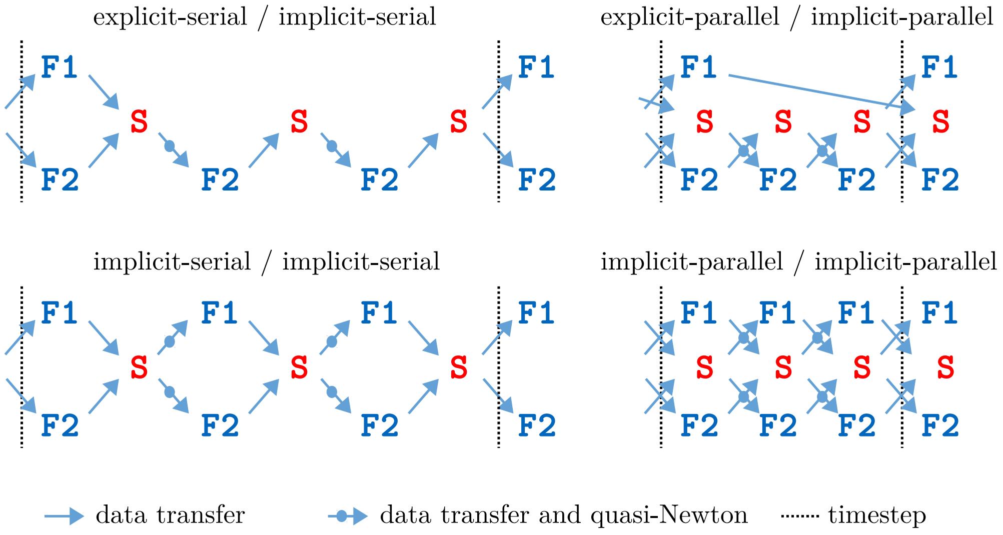

explicit-serial / implicit-serial
explicit-parallel / implicit-parallel
implicit-serial / implicit-serial
implicit-parallel / implicit-parallel
F1
S
F2
data transfer
data transfer and quasi-Newton
timestep

Figure 48: Fluid-structure-fluid model problem: execution orders for various combinations of two bi-coupling schemes. For the sake of clarity, I assume that every coupling scheme converges in three iterations.

well as $F2$-S via past input-output information. The structure operator S changes in the meantime, which renders these past input-output information invalid. This marks an inconsistency. The same reasoning holds for a combination of two parallel-implicit schemes or a mixed serial-implicit parallel-implicit variant. This affects not only quasi-Newton schemes, but also, e.g., an Aitken underrelaxation, since also here past information is used to construct an optimal relaxation factor. To which extent this inconsistency concerns the robustness of such a composed multi-coupling depends assumably on the strength of the indirect interaction between both fluid fields. The next section confirms this assumption by showing a strong influence of the structure's density ρ_S and width δ. In general, this problem appears as soon as there is at least one strong indirect interaction. Thus, a composition of bi-coupling might be a valid strategy for some cases, but is no general solution.

Inclusion of Bi-Coupling Schemes A solution to the inconsistency problem of the composition, while still reusing bi-coupling schemes, is an inclusion of such schemes. Without loss of generality, the bi-coupling $F1 - S$ is regarded as an entity from the outside and coupled as a whole to $F2$. Figure 49 visualizes this approach. This inclusion introduces a nesting of coupling schemes: the inner schemes is iterated till convergence, and only then the outer scheme performs one iteration. If one of both schemes is an explicit scheme, this results in a useful overall scheme. Primarily, however, this is only another theoretical formulation of similar composition schemes. An inner implicit-serial scheme included in an outer explicit-parallel schemes, for example, is identical to an explicit-serial/implicit-serial composition, compare Figure 48, top row, left. On the other side, if both schemes, inner and outer, are chosen to be implicit, past input-output information remains always consistent. The nesting, however, leads to a drastic increase of iterations. If more implicit schemes are nested, the number of iteration grows exponentially with the number of schemes [44]. Therefore, such an inclusion of bi-coupling schemes is of no practical relevance and, thus, not implemented in preCICE.

True Multi-Coupling As already indicated briefly in the introduction of this section, a generalization of the idea that leads from (GS) to (J), results in an overall robust multi-coupling scheme. All three physical field are executed at the same time, all outputs are concatenated in a single vector and serve as input for the next iteration. This results in the overall fixed-point equation

$$(F1, S, F2)(d_1, d_2, f_1, f_2) = (d_1, d_2, f_1, f_2) \ . \qquad \text{(M)}$$

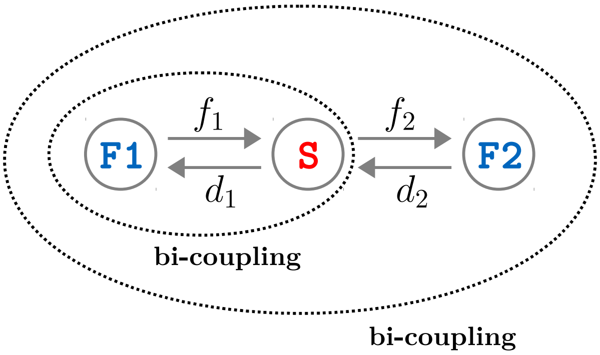


Figure 49: Fluid-structure-fluid model problem: inclusion of bi-coupling schemes.

This notation allows for a straightforward generalization to other multi-coupling scenarios. The single dependencies between individual solvers and variables, however, are hidden. I refer to such schemes by, e.g., M-AA(5), if the fixed-point equation is solved by an Anderson acceleration and, as usual, columns from five previous timesteps are reused. (M), of course, suffers from the same possible in-balance as (J), compare Section 3.6.3. Therefore, I apply a similar solution, namely, the residual-sum weighting, throughout all numerical experiments of the next section.

3.8.3 Numerical Experiments

This section contains numerical results for both testcases, FSF and F4S. First, the FSF model problem is discussed in detail. Afterwards, the F4S testcase is introduced and serves as further validation. Finally, I conclude on best practices.

FSF Problem Both fluid solvers are simulated with Alya Nastin, the structure solver with Alya Solidz. Both fluid domains consist of 2600 elements. Depending on the structural width $\delta = 0.1\,\text{m}$, $0.2\,\text{m}$, or $1.0\,\text{m}$, the structure domain consists of 200, 206 or 1306 elements, respectively. All meshes always match at the interfaces. The dynamic viscosity of both fluids is $\mu = 100\,\text{kg}/(\text{m}\,\text{s})$. The Young's modulus of the structure is $E = 5.6 \times 10^6\,\text{N}/\text{m}^2$ and the Poisson ratio $\nu = 0.4$. Figure 50 shows physical results for $\delta = 0.1\,\text{m}$ and $\rho_{F1} = 1.0 \times 10^2\,\text{kg}/\text{m}^3$, $\rho_S = \rho_{F2} = 1.0 \times 10^3\,\text{kg}/\text{m}^3$. Throughout the first five periods, no periodic movement of the structure establishes, but a continuous deformation. As FSI stopping criterion, I choose, as usual, a relative criterion of 10^{-4} for all involved coupling variables, here f_1, f_2, d_1, d_2. A QR1 filter with $\epsilon = 10^{-6}$ is applied. Table 22 lists the average number of iterations for all three structural widths δ and various density combinations. For each configuration, I compare two composition schemes GS-AA(10) / GS-AA(10) and J-AA(20) /J-AA(20) with two overall multi-coupling schemes M-AA(20) and M-GB(0). Furthermore, results for the explicit-implicit combination J-EX / J-AA(20) are shown.

Several tendencies can be observed. First, an explicit-implicit combination can be useful if one of both interaction is very weak, but not the other one. If the first grows in strength, however, such a scheme tends, of course, to be unstable due to the added-mass effect. The first two lines of Table 22 illustrate this. Next, a combination of two implicit schemes can work out, if the indirect interaction between both fluid fields is weak. This is the case if both fluid densities are equal and the structure density is not too small or for a bigger structural width δ. In general, a combination of two (J) schemes tends to be more robust than a combination of two (GS) schemes. Finally, both overall multi-coupling approaches appear very reliable and efficient. The number of iterations raises, as expected, with the difficulty of the scenario, similar to the instability of the combination schemes. M-AA(20) and M-GB(0) show, in general, very similar results, whereas as the latter tends to be a little bit more efficient for this testcase.

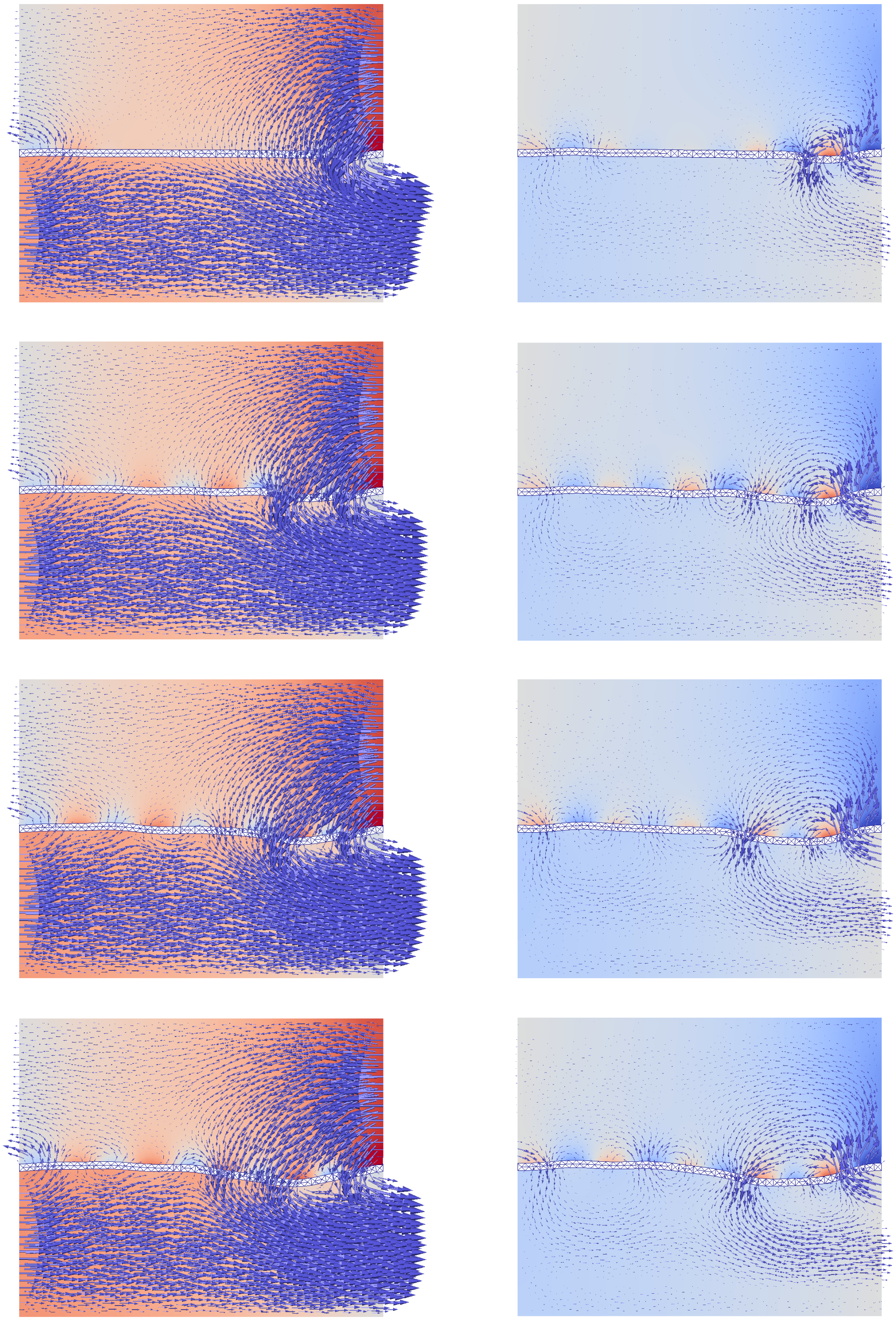


Figure 50: Fluid-structure-fluid model problem: pressure values (scaled from $1.0 \times 10^{-4}\,\mathrm{Pa}$ to $1.0 \times 10^{4}\,\mathrm{Pa}$), velocity vectors and structure deformation, from left to right and from top to bottom at $t = i \cdot 0.05\,\mathrm{s}$, $i = 1 \ldots 8$. $d = 0.1\,\mathrm{m}$, $\rho_{F1} = 1.0 \times 10^{2}\,\mathrm{kg/m^3}$, $\rho_S = \rho_{F2} = 1.0 \times 10^{3}\,\mathrm{kg/m^3}$.

ρ_{F1} - ρ_S - ρ_{F2}	Coupling	$\delta = 0.1\,\mathrm{m}$	$\delta = 0.2\,\mathrm{m}$	$\delta = 1.0\,\mathrm{m}$
10^0 - 10^3 - 10^3	J-EX / J-AA(20)	1.00 / 16,85	1.00 / 12.56	1.00 / 8.68
10^1 - 10^3 - 10^3	J-EX / J-AA(20)	*div*	*div*	*div*
10^0 - 10^3 - 10^3	GS-AA(10) / GS-AA(10)	*div*	*div*	3.71 / 7.06
10^1 - 10^3 - 10^3	GS-AA(10) / GS-AA(10)	*div*	*div*	3.02 / 7.09
10^2 - 10^3 - 10^3	GS-AA(10) / GS-AA(10)	*div*	*div*	3.21 / 6.63
10^3 - 10^3 - 10^3	GS-AA(10) / GS-AA(10)	*div*	*div*	6.23 / 6.33
10^3 - 10^2 - 10^3	GS-AA(10) / GS-AA(10)	*div*	*div*	*div*
10^3 - 10^1 - 10^3	GS-AA(10) / GS-AA(10)	*div*	*div*	*div*
10^0 - 10^3 - 10^3	J-AA(20) / J-AA(20)	9.38 / 17.80	4.82 / 12.16	4.29 / 9.03
10^1 - 10^3 - 10^3	J-AA(20) / J-AA(20)	*div*	2.82 / 12.23	2.10 / 9.32
10^2 - 10^3 - 10^3	J-AA(20) / J-AA(20)	9.05 / 16.41	2.52 / 9.42	2.24 / 7.71
10^3 - 10^3 - 10^3	J-AA(20) / J-AA(20)	7.25 / 9.29	7,19 / 6.95	5.67 / 6.22
10^3 - 10^2 - 10^3	J-AA(20) / J-AA(20)	*div*	*div*	6.95 / 7.05
10^3 - 10^1 - 10^3	J-AA(20) / J-AA(20)	*div*	*div*	*div*
10^0 - 10^3 - 10^3	M-AA(20)	13.37	11.67	9.62
10^1 - 10^3 - 10^3	M-AA(20)	11.33	9.77	8.38
10^2 - 10^3 - 10^3	M-AA(20)	10.39	7.71	7.45
10^3 - 10^3 - 10^3	M-AA(20)	9.01	7.33	6.71
10^3 - 10^2 - 10^3	M-AA(20)	10.96	8.63	8.46
10^3 - 10^1 - 10^3	M-AA(20)	14.07	9.86	9.81
10^0 - 10^3 - 10^3	M-GB(0)	12.35	8.18	9.04
10^1 - 10^3 - 10^3	M-GB(0)	10.35	8.04	7.66
10^2 - 10^3 - 10^3	M-GB(0)	8.62	6.78	6.48
10^3 - 10^3 - 10^3	M-GB(0)	7.73	6.36	6.41
10^3 - 10^2 - 10^3	M-GB(0)	9.89	7.34	7.02
10^3 - 10^1 - 10^3	M-GB(0)	12.98	8.61	8.33

Table 22: Average number of iterations for various density settings and three different structural widths δ. Two implicit combination approaches are compared to two overall multi-coupling schemes. Furthermore, results for an explicit-implicit combination are shown. *div* marks divergent configurations. For the combination approaches, the number of iterations are listed for every interaction individually, separated by a slash.

F4S Problem To further validate the multi-coupling schemes, I consider another testcase. Four elastic structures are immersed in a channel flow. Each structure has a different density, making the overall scenario quite challenging. Figure 51 depicts the geometry. The densities read:

F	$S1$	$S2$	$S3$	$S4$
$1.0\,\mathrm{kg/m^3}$	$1.0 \times 10^1\,\mathrm{kg/m^3}$	$1.0 \times 10^{-1}\,\mathrm{kg/m^3}$	$1.0\,\mathrm{kg/m^3}$	$1.0 \times 10^2\,\mathrm{kg/m^3}$

The Young's modulus of the structure is $E = 5.0 \times 10^5\,\mathrm{N/m^2}$ and the Poisson ratio $\nu = 0.4$. The dynamic viscosity of the fluid is $\mu = 1.0\,\mathrm{kg/(m\,s)}$ and the timestep size $\delta t = 1 \times 10^{-4}\,\mathrm{s}$. The Alya solvers Nastin and Solidz are used for all fields. The fluid domain is discretized with 4702 elements, whereas each structure domain consists of 140 elements. All coupling interfaces match. At the inflow, a constant parabolic velocity profile with mean 50.0 m/s is prescribed. Figure 52 shows physical results.

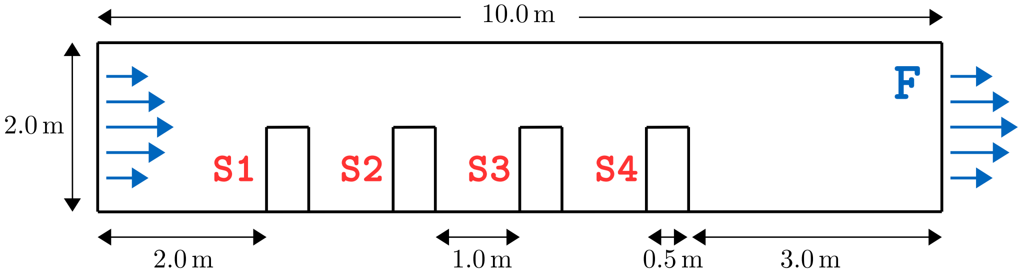


Figure 51: F4S multi-flap testcase: geometry sketch.

As FSI convergence measure, all sub-vectors are checked with a relative criterion of 10^{-4}. A QR1 filter with $\epsilon = 10^{-6}$ is applied. All combination schemes fail to converge for this testcase as they cannot handle the global interaction complexity. The overall multi-coupling, however, shows again a very robust behavior. The M-AA(20) scheme needs on average 6.44 iterations to converge during the first 200 timesteps, whereas the M-GB(0) schemes requires 8.34 iterations.

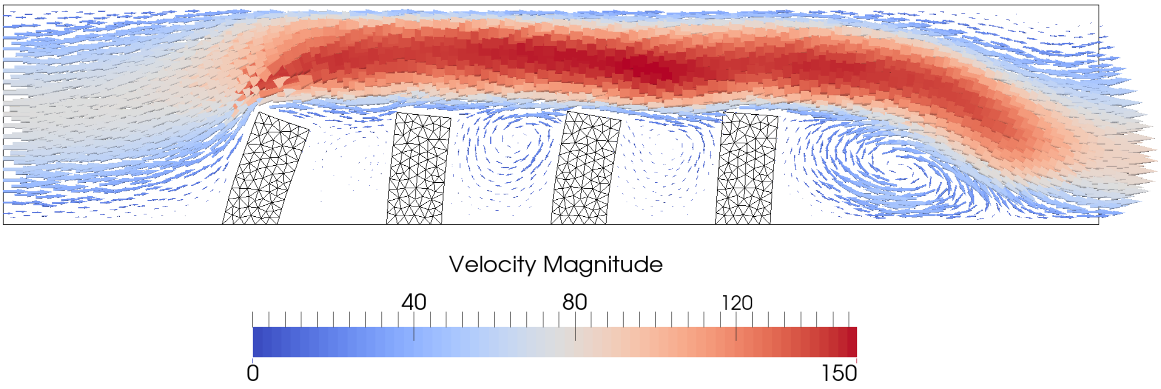


Figure 52: Multi-flap F4S testcase: velocity vectors and structure deformation at $t = 0.03\,\mathrm{s}$.

Conclusion I want to stress the most important results again. A combination of bi-coupling schemes is a reasonable way to go if a strong interaction is combined with several weak interactions. Combing then an implicit scheme with several explicit ones is the right choice. If more than one strong interaction is present, combination schemes can still work, but are far from being robust. In such case, an overall multi-coupling approaches tends to be a very good choice. Both variants, either with an Anderson acceleration or with a generalized Broyden solver show good and robust results.

Summary of Chapter 3

- State-of-the-art sequential coupling schemes, such as the IQN-ILS scheme, are a combination of a sequential block-Gauß-Seidel-like fixed-point equation (GS) and sophisticated fixed-point equation solvers.
- I introduce two fixed-point equations that allow for a simultaneous execution of the fluid and the structure solver: a block-Jacobi one (J) and one referred to as Steklov-Poincaré (SP).
- Sophisticated fixed-point equation solvers can be interpreted as multi-secant methods: they approximate the Jacobian of the residual system by a multi-secant equation.
- To get to a unique approximation, I consider two possibilities: The Anderson acceleration (AA) minimizes the norm of the Jacobian itself while the generalized Broyden method (GB) minimizes the distance to the approximation from the last timestep.
- AA allows for a matrix-free implementation, but can reuse information from previous timesteps only in an explicit fashion, resulting in a tuning parameter. GB requires the storage of the complete Jacobian, but reuses previous information implicitly.
- Both methods come in two flavors: type I directly approximates the Jacobian, while type II approximates the inverse Jacobian.
- I compare all combinations of fixed-point equations and multi-secant methods by means of a simple 1D FSI example.
- (J) and (SP) as well as AA and GB show similar performance. Type II methods are more robust than type I methods.
- I conclude to implement (J) besides (GS) and both multi-secant methods in type II in preCICE.
- The implementation clearly separates the fixed-point equation solvers (post-processing schemes in preCICE nomenclature) from the fixed-point equations. Thereby, arbitrary combinations are possible. I refer to such combinations also as quasi-Newton schemes.
- A filtering of the column space of AA and GB improves the methods' robustness and efficiency.
- (J) requires a weighting of its sub-vectors to handle possibly different scales. A weighting by the sum of the squares of the residuals from the current timesteps (residual-sum weighting) appears to be an efficient and robust choice.
- I further compare all in preCICE implemented schemes by means of three FSI benchmark scenarios. In general, (J) shows indeed a similar convergence speed as (GS). AA and GB also show a similar performance, while GB needs no explicit reuse of information from previous timesteps. All quasi-Newton schemes significantly outperform the classical Aitken underrelexation.
- (J) can be generalized to include more than two solvers. Such a multi-coupling scheme allows for a robust coupling of scenarios that involve more than one strong interaction. A simple composition of classical schemes fails for such cases.

4 Intra-Solver Parallelism: preCICE on Distributed Data

In the introduction of this thesis, Section 1.3.2, I motivate why porting preCICE to distributed data and avoiding, thereby, any central instance is a key ingredient to allow for massively parallel multi-physics simulations. This chapter explains the concepts, the implementation and the performance of the parallelization of preCICE. Chapter 2 is a prerequisite for this chapter. Please recall Figure 8 on page 11, which visualizes the new distributed layout of preCICE. Porting preCICE to this layout, comes with several challenges. First, a steering concept needs to be realized and communicated surface meshes need to be re-partitioned on the receiver side according to the receiver's partitioning and the used interpolation methods. Section 4.1 details these two topics. Then, the three main feature groups need to be ported to distributed data. Section 4.2, Section 4.3, and Section 4.4 explain the realization of the communication, the interpolation methods, and the coupling schemes on distributed data. Section 4.1 is a prerequisite for all three Sections 4.2 to 4.4. These three sections, however, can be read independently of each other.

I favor the expression *porting to distributed data* over the term *parallelization*, since the main goal is not a classical speed-up, but rather rendering a central instance unnecessary by a fully parallel peer-to-peer concept. I, therefore, do in general not consider every single performance delta. More important is that preCICE does not degenerate the overall scalability of a coupled simulation. preCICE solely operates on surface data. On the one hand, this limits the scalability of the coupling operations itself, since the compute effort is limited and often dominated by communication. On the other hand, this makes the coupling operations cheap and, thus, easy to hide behind compute-intensive single-physics solvers. I motivate in Section 1.3.2 that optimizing the work per timestep, meaning the time spent in `advance`, is of high importance, while the initialization effort, the time spent in `initialize`, should remain tolerable. For the latter, Section 1.3.2 defines tolerable as roughly 10 seconds. I follow these two guidelines throughout this chapter.

The sections of this chapter all share a similar outline. I always start with a simple algorithmic description, which is followed by a more technical description, focusing on the implementation and on the parallelization. Finally, I discuss scalability for each section, and thus, for each feature group, separately. For the latter, I use the Artificial Solver Testing Environment (ASTE), which I explain after this brief introduction. After the separate discussion of all feature groups, Section 4.5 shows the scalability of two complete simulations to study the interplay of all parts of preCICE with each other as well as the influence of preCICE on the overall performance.

As preCICE is a joint software project, also the parallelization is a common effort. The parallel communication was developed as part of the master thesis of Alexander Shukaev [181]. On the parallelization of the radial basis function mappings, I worked together with Florian Lindner. The parallelization of the quasi-Newton coupling schemes was a common effort with Klaudius Scheufele. Parts of the parallelization are already published in [43]. The explanations of this chapter also follow this publication to some extent. The first physical scaling experiment, the Ateles Cube, is joint work with Verena Krupp. We published preliminary results for this case in [41]. The second physical scaling experiment, the PfS-1a benchmark [58] is a joint effort with Juan-Carlos Cajas and Herbert Owen. Guillaume de Nayer provided us with the original meshes of the benchmark, which I want to acknowledge thankfully.

Like already stated above, the introduction to this chapters ends with a brief presentation of ASTE. Furthermore, I introduce the testing hardware. The fast reader may directly jump to Section 4.1 as theses two paragraphs only introduce obvious notation besides technical details.

Artificial Solver Testing Environment ASTE is the simplest way to test parallelized implementations in preCICE. It is nothing more than an artificial preCICE proxy, which only defines an interface mesh, writes data to it and reads data from it. ASTE itself is open source[40] to enable the reader to redo the scaling experiments. In principle, ASTE operates on the preCICE API level. I use it, however, only to test the parallel performance on a package level. Like stated above, in Section 4.5, I use real physical solvers to test the scalability also on the API level. ASTE defines a 2D equidistant Cartesian mesh with $\hat{n}$ vertices along each coordinate axis. This sums up to $n = \hat{n}^2$ vertices. The side length in each direction is 1. Please note that this mesh corresponds to an interface mesh for an actual physical simulation,

[40] `https://github.com/precice/aste`

hence a lower-dimensional manifold. The consequences of this must be considered when interpreting scalability results. For the sake of simplicity, the mesh coordinates are 2D, however, not 3D. The surface mesh is, thus, a plane instead of a real manifold. To decompose the mesh equally among p cores, the mesh is linearized and ordered linewise, like depicted in Figure 53. p refers to the number of cores per participant throughout this thesis. To test preCICE, always two ASTE participants are executed. All implementations described in this chapter, however, are fully functional for scenarios involving more than two participants, compare, for example, Section 5.5. I refer to the two participants of the ASTE tests by A and B. Whereever necessary, I add an index A or B to any variables to distinguish between both.

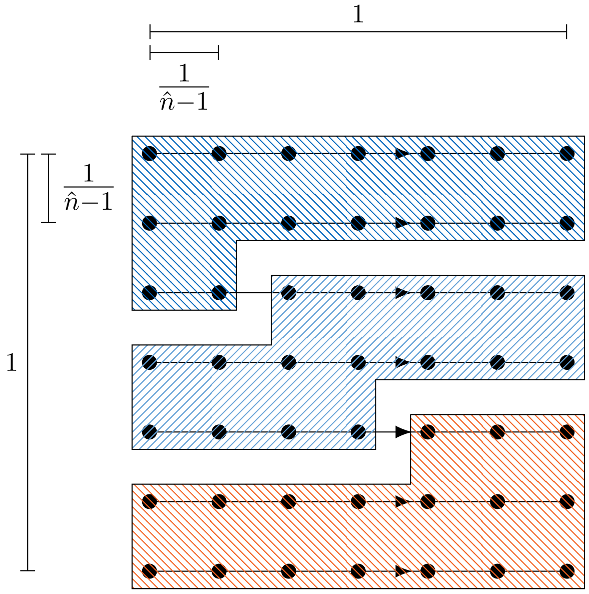


Figure 53: Exemplary mesh decomposition of ASTE. Vertices are distributed along a linewise linearization among three processors. Arrows indicate the linewise ordering. $\hat{n} = 7$. A similar figure was already used in [43].

Hardware and Time Measurement All performance tests of this thesis were executed on SuperMUC, hosted at the Leibniz Supercomputing Center in Garching. All ASTE tests and the Pfs-1a scaling experiment in Section 4.5.2 use phase 2 of SuperMUC, holding Haswell Xeon processors of type E5-2697 v3. Such nodes consist of 28 cores and are interconnected with a FDR14 Infiniband. The Ateles Cube scaling experiment in Section 4.5.1 uses the thin nodes partition of phase 1 of SuperMUC, holding Sandy Bridge-EP Xeon E5-2680 8C processors, with 16 cores per node, interconnected via a FDR10 Infiniband[41]. If not stated differently, all runs always use full nodes. For the time measurement, I use a built-in event framework in preCICE. A single event has a time resolution of 1ms, which is sufficient considering the high costs of a typical single-physics timestep. Also, if not stated differently, I always perform five runtime experiments, from which I drop the minimum and the maximum. Afterwards, the three remaining experiments are averaged. Furthermore, I always run ten timesteps with five coupling iterations each. Thus, the work per `advance` is further averaged over 50 calls. I only list the variance of single events if it is significant. Finally, I apply an artificial synchronization strategy to avoid diluting the measurements of single events by a non-perfect load-balancing of preceding events. Therefore, I synchronize all ranks of a single participant before and after each intra-participant event and the two master ranks before each inter-participant event.

[41] For more technical details, see `https://www.lrz.de/services/compute/supermuc/systemdescription/`.

4.1 Steering Concept and Re-Partitioning of Meshes

The old server-based concept of preCICE [99] distinguishes between the `client mode`, the `server mode`, and the `coupling mode`. A serial participant runs in `coupling mode`. For a parallel participant, every rank runs in `client mode`, while a server needs to be started, which then runs in `server mode` and `coupling mode`. The new fully-parallel concept of preCICE is realized via two further modes: the `master mode` and the `slave mode`. For every participant, rank 0 runs in `master mode`, whereas all other ranks run in `slave mode`. This choice is rather arbitrary and can be adapted easily. Furthermore, it is hidden from the user, meaning that from the user's perspective each rank appears similar. This includes, for example, the fact that the master can also hold a part of the mesh. During the configuration of preCICE, the master-slave communication is established in a generic way. Any existing 1:N communication can be configured, completely independent from the solvers communication. Compare Section 2.1.2 or Section 4.2 for more information on communication in preCICE. MPI communication is the most efficient choice, since optimized routines for reduction or broadcasts are provided. However, also a TCP/IP communication can be used to avoid any MPI dependency for binary distributed closed-source solvers.

To briefly recapitulate Chapter 2: the two main steering methods of preCICE are `initialize` and `advance`. `initialize` needs to be called once at the beginning of a simulation to, among other things, communicate coupling meshes and set up the communication. After every timestep, the user calls `advance` to apply interpolation methods and equation coupling schemes and to communicate data. Steering between the master rank and the slave ranks can be reduced to a minimal effort during `advance`. For explicit coupling, no sychronization at all is needed. For implicit coupling, convergence measures need to be computed. This reduces to a simple reduction and broadcast of floating point values. Of course, sophisticated interpolation schemes, such as radial basis function mappings, or equation coupling with quasi-Newton schemes need further synchronization during `advance`. Sections 4.3.2 and 4.4.2 give more details. Furthermore, as depicted in Figure 8, coupling meshes and associated data fields are stored locally. Reading and writing to and from the mesh does not need any communication, contrary to the old server-based concept. To sum up, the new fully-parallel concept only needs minimal communication between the ranks of a single participant during one timestep. This comes however at the cost of significant work during `initialize`. I motivated in Section 1.3.2 that this effort does not need to be highly efficient, but needs to remain tolerable to allow for massively parallel simulations. This section details a key initialization step: the re-partitioning of communicated coupling meshes. Therefore, Section 4.1.1 gives an algorithmic description while Section 4.1.2 focuses on the implementation in preCICE. Afterwards, Section 4.1.3 gives performance results.

4.1.1 Algorithmic Description

Let us consider that each participant defines one coupling mesh. As mentioned just above, coupling meshes are defined locally, hence

$$\bigcup_{i=0}^{p_A-1} \Gamma_A^i = \Gamma_A \ , \quad \bigcup_{i=0}^{p_B-1} \Gamma_B^i = \Gamma_B \ ,$$

where Γ_A^i, Γ_B^i denote the local components and Γ_A, Γ_B the global meshes. A mesh is nothing else but a set of vertices. Optionally, a mesh can also hold connectivity information such as edges or triangles. Such connectivity information is, however, only needed for rare cases in preCICE as, for example, the nearest projection mapping. For the sake of simplicity, I neglect such cases here. The local mesh parts are not necessarily disjoint. Depending on the domain decomposition of the solver, the user might define the same vertex on multiple ranks. preCICE handles such cases by treating identical vertices as different vertices with the same coordinates. The users needs to be aware of the consequences. A nearest neighbor mapping, for example, then finds an arbitrary representative of those duplicated vertices. While this might be a valid approach for certain cases, it is recommended, in general, to define duplicated vertices only once. For a standard solver, this is typically no restriction.

In order to map values from Γ_A to Γ_B, one of the two meshes needs to be sent to the other participant. Without loss of generality, Γ_B is sent from B to A and re-partitioned there. I denote the re-partitioned local mesh by $\tilde{\Gamma}_B^i$. Then, the user can define a mapping between both meshes at participant A (compare

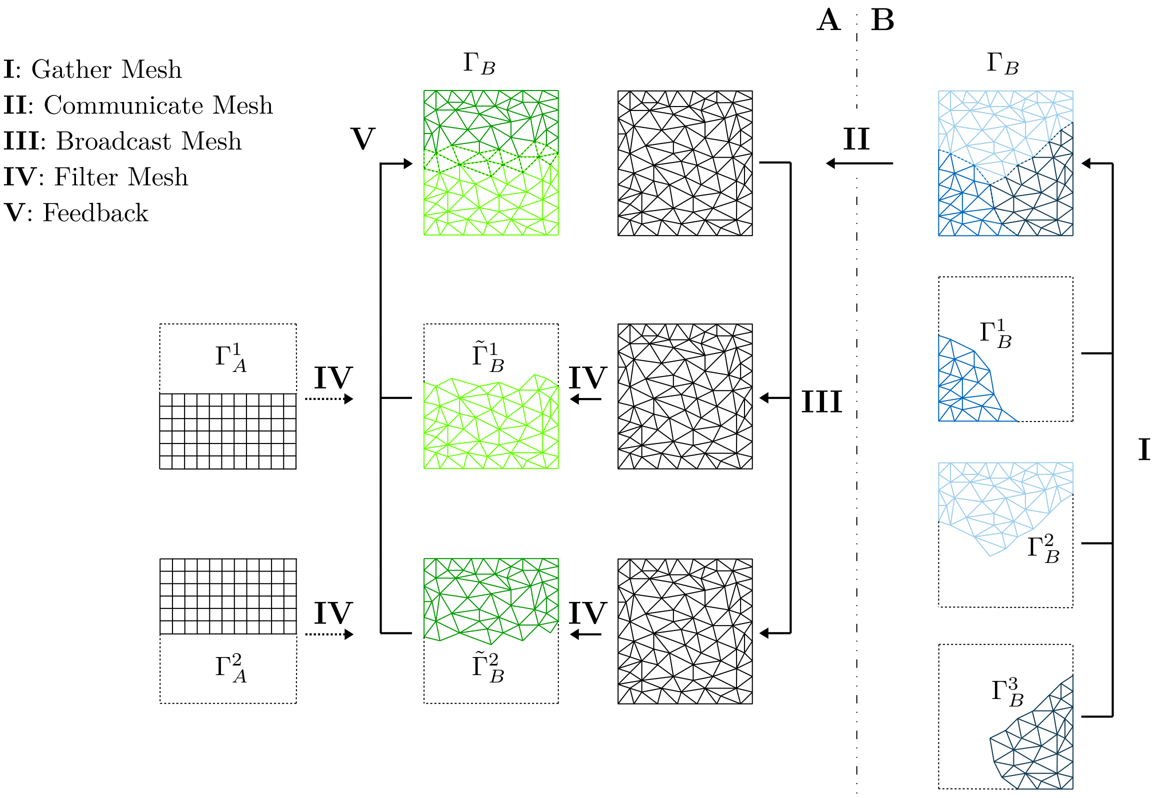


Figure 54: Re-partitioning of meshes: *broadcast/filter* variant. Participants A and B run on three and four processors, respectively. Each participant defines local mesh contributions, depicted in black for A, on the left, and in blue for B, on the right. Γ_B is sent from B to A to allow for a mapping between Γ_B and Γ_A at participant A. Therefore, $\Gamma_B = \cup_{i=0}^{3}\Gamma_B^i$, is gathered at the master of B (step I) and then sent to the master of A (step II). Afterwards, Γ_B is broadcast on A and filtered by each slave locally (step IV), resulting in a re-partition $\Gamma_B = \cup_{i=0}^{3}\tilde{\Gamma}_B^i$, depicted in green. Finally, information on the filtering is fed back to the master of A (step V). For the sake of simplicity, both masters do not hold an own mesh partition in this case. Also, slaves could hold empty partitions. Furthermore, optional connectivity information is visualized in this sketch.

Section 4.3) and data communication between both participants via Γ_B (compare Section 4.2). I consider two variants for re-partitioning: *broadcast/filter* and *pre-filter/post-filter*. Figure 55 lists both variants, whereas Figure 54 visualizes the *broadcast/filter* approach. By filter, I denote the concept of removing vertices from the global mesh that have no influence on a local partition. In step **IV**, for example, both variants apply a mapping-dependent filter. For projection-based mappings, this is realized by the computation of a preliminary mapping and the withdrawal of all vertices that do not influence the mapping. This filter can be preceded by a further bounding box filter to speed up the mapping. For a radial-basis function based interpolation, a filtering step according to the support radius of the basis function is applied. Section 4.3.2 gives more details.

Discussion The *pre-filter/post-filter* variant features a serial part in the pre-filter step: the master of A treats one slave after the other. Therefore, the complexity raises linearly with n_B and linearly with p_A. The total communicated data is $O(n_B)$. If the master-slave communication is implemented via MPI, the *broadcast/filter* variant can rely on sophisticated MPI broadcast routines, which cost $O(n_B \log(p_A))$. If the TCP/IP implementation is used, however, such optimized algorithms are currently not supported, which leads the complexity degenerate to $O(n_B p_A)$. The filter step for the *broadcast/filter* variant, rises linearly with n_B, whereas the post-filter step for the *pre-filter/post-filter* approach only needs to filter a local mesh, which reduces the computational effort. To conclude, the *broadcast/filter* variant is in general favorable, especially for a higher number of cores. The *pre-filter/post-filter* might, however, be preferable for a huge mesh Γ_B and an only moderate number of cores or for a case where an MPI communication is technically not possible.

	broadcast/filter		**pre-filter/post-filter**
I	M_B gathers Γ_B.	**I**	M_B gathers Γ_B.
II	M_B sends Γ_B from B to A.	**II**	M_B sends Γ_B from B to A.
IIIa	M_A broadcasts Γ_B to each S_A^i (*broadcast*).	**IIIb**	Every S_A^i sends a bounding box around Γ_A^i to M_A. M_A filters Γ_B accordingly and sends the filtered mesh to S_A^i (*pre-filter*).
IVa	Every S_A^i filters the mesh according to the defined mappings (*filter*).	**IVb**	Every S_A^i filters the mesh again according to the defined mappings (*post-filter*).
V	M_A gathers distribution information from all S_A^i.	**V**	M_A gathers distribution information from all S_A^i.

Figure 55: Mesh re-partitioning strategies. The two strategies only differ in step III and IV. S_A^i denotes a single slave rank from participant A, M_A, M_B the master ranks of participants A and B, respectively.

4.1.2 Implementation

The mesh-repartitioning is implemented in the `geometry` package. Please recall the overview in Table 1 on page 23 and Figure 20 on page 22. Figure 56 briefly sketches the software architecture of the `geometry` package, including the newly added components. In preCICE nomenclature, a geometry is a layer around a mesh, which indicates how the mesh is created. For a `SolverGeometry`, the mesh is created by the associated participant, hence by the user. For a `CommunicatedGeometry`, the mesh is sent or received by the associated participant. For an `ImportedGeometry`, the mesh is read from a file. For more details, please refer to [99]. The newly developed mesh-repartitioning strategies need, hence, be solely applied for received `CommunicatedGeometries`.

Compared to the original preCICE version, the order in which the geometries create the meshes needs to be adapted if more than one coupling mesh is communicated. Both participants, A and B, need to communicate all such meshes in the same order to avoid deadlocks, since the asynchronous communication still becomes synchronous for large meshes. Afterwards, however, a re-ordering of the geometries for the re-partitioning is necessary. Since the mesh re-partitioning strategies depend on the mappings associated with the received mesh, all `SolverGeometries` need to create their meshes before the received meshes can be re-partitioned.

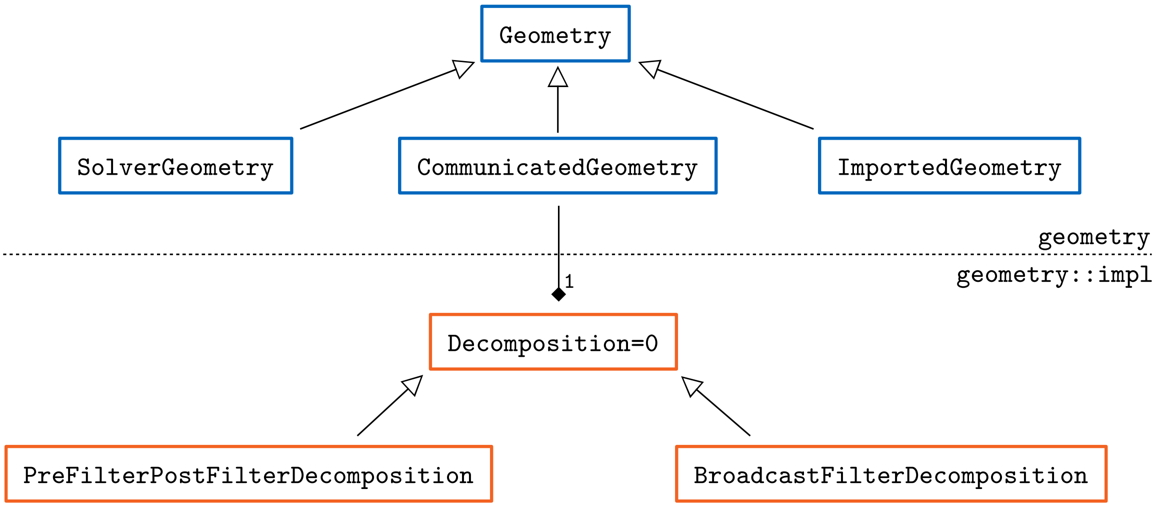


Figure 56: Software architecture of the `geometry` package. The arrow notation follows the UML standard (empty triangles: inheritence, filled diamonds: containement). Original components [99] are marked in blue, whereas new components are marked in orange. Mesh re-partitioning is referred to as `Decomposition` in preCICE nomenclature.

4.1.3 Scaling Results

The scalability tests encompass three strong scaling series for $n = 448^2$, $n = 896^2$, and $n = 1792^2$. The number of cores per participant is doubled four times from $p = 112$ to $p = 1792$. I use the MPI implementation for the intra-participant communication and TCP/IP sockets for the communication between both participants. Section 4.2 gives more information on the communication. Between both meshes, nearest-neighbor mappings are used. The goal of the scalability study of the mesh-repartitioning is not to resolve and analyze every memory hierarchy feature, but rather to get an overall impression on this initialization step. This means up to which mesh size do these non-optimized re-partitioning strategies remain tolerable and how are they influenced by the number or cores. Figures 57, 58, 59 visualize the results for the series, respectively. I compare both strategies, *broadcast/filter* and *pre-filter/post-filter*. Since step I and II are identical for both approaches, they are listed only once. Whereas step III and IV differ essentially, step V is identical for both strategies. Still, step V is again listed twice, as it operates on different data.

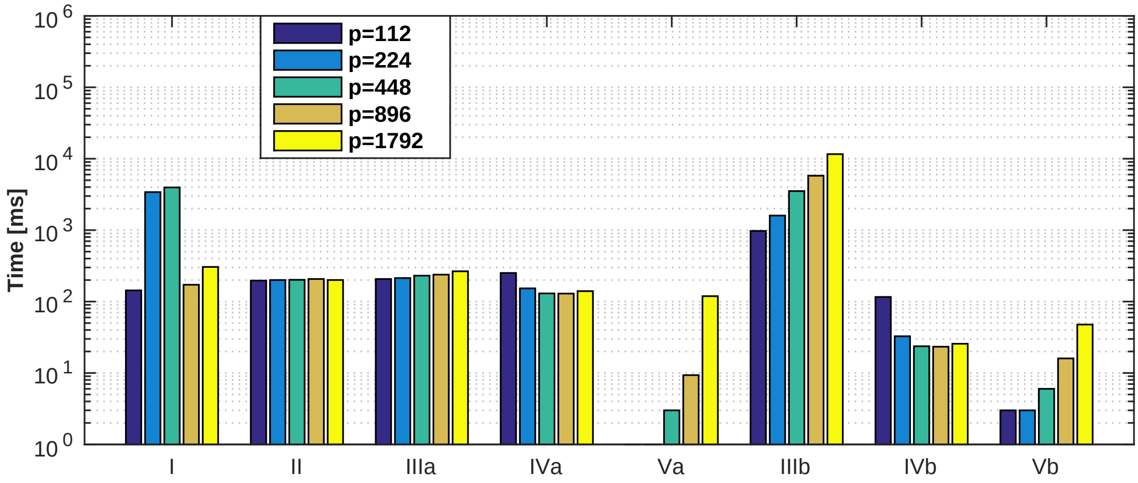


Figure 57: Mesh re-partition: strong scaling results for ASTE with $n = 448^2 \approx 2.0 \cdot 10^5$ vertices. The *broadcast/filter* approach (a) is compared to the *pre-filter/post-filter* approach (b).

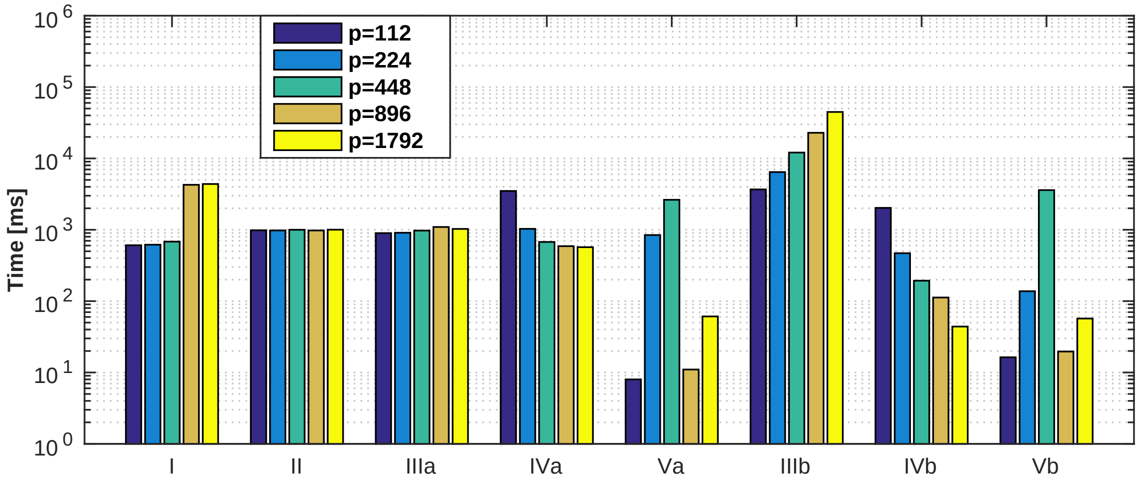


Figure 58: Mesh re-partition: strong scaling results for ASTE with $n = 896^2 \approx 8.0 \cdot 10^5$ vertices. The *broadcast/filter* approach (a) is compared to the *pre-filter/post-filter* approach (b).

In the following, I first analyze every single step separately, focusing on the influence of n and p. Afterwards, I comment on the relation between the steps and on the overall comparison of both approaches. In step I, the mesh is gathered at the master rank of B. The total amount of communicated data is not

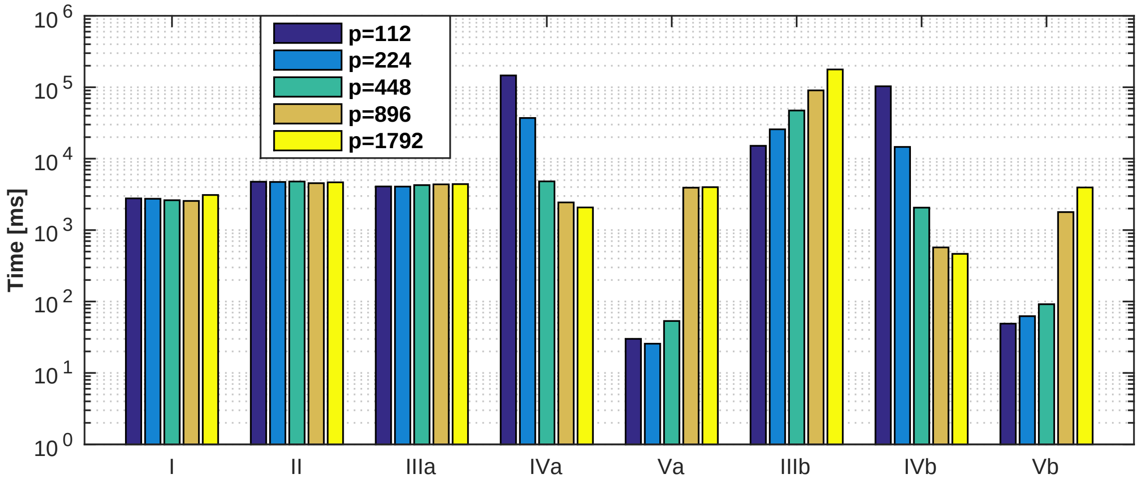


Figure 59: Mesh re-partition: strong scaling results for ASTE with $n = 1792^2 \approx 3.2 \cdot 10^6$ vertices. The *broadcast/filter* approach (a) is compared to the *pre-filter/post-filter* approach (b).

dependent on p, only the decomposition changes. For $n = 448^2$ and $n = 896^2$, there is a pike visible, which stems from the overlapping of receive operations and the actual mesh creation, meaning reservation of heap memory at the master rank. If the chunks per slave rank are sufficiently large, the overhead for accessing memory is minimal, such that the pike disappears. On the other hand, for sufficiently small chunks, there is a memory hierarchy benefit visible, which also effects other mesh re-partitioning steps. Except this pike, the runtime is rather independent of p and rises linearly with n. Please note that the overlapping of receiving mesh data and the mesh creation leaves room for improvement. Also the usage of `MPI_Gather` should result in a speed-up. Still, step I is, in general, a cheap operation, such that further analysis and optimization is not necessary at the moment. Step II simply sends the complete mesh from the master rank of B to the master rank of A. This operation does not depend on p and rises, as expected, linearly with n. The same overlapping routine as for step I is used, hence the same non-optimal performance. However, again, the overall costs are tolerable.

For the *broadcast/filter* strategy, step IIIa broadcasts the complete mesh from the master rank of A to all ranks of A. While the theoretical logarithmic dependence on p is hardly visible, the costs increase linearly with n. Step IVa filters the complete mesh locally on every rank, first via a bounding box filter, then via the pre-liminary mappings. While the first filter scales linearly with n, the second one scales quadratically with n/p. For $n = 448^2$, the first part dominates, hence the rather constant runtime in p. For $n = 896^2$ and $n = 1792^2$, the second part dominates for a smaller p, hence a quadratic speed-up with p, while the first one dominates for a higher p. This, of course, also influences the dependence on n. If the first part dominates, a linear dependence on n is visible. However, if the second part dominates the overall costs for the filtering step, i.e. large n and small p (e.g. $n = 1792^2, p = 112$), IVa increase quadratically with n, which can, thus, be very costly. Step Va feeds the re-partition information back to the master rank of A. This marks a serial step at the master rank, though the overall amount of communicated data remains constant. For $n = 896^2$, there is a drop of the runtime visible from $p = 448$ to $p = 896$, due to the send operation of the slave ranks, which becomes asynchronous for smaller chunk sizes. Except this drop, the master's overhead per rank lets the runtime increase with p.

For the *pre-filter/post-filter* strategy, step IIIb pre-filters and communicates the mesh for every rank individually. This leads to a serial step which increases linearly with p. Afterwards, step IVa post-filters locally via the preliminary mappings. As this then only features local meshes, the scalability is quadratic in p, overlaid with a cheap serial overhead. Finally, step Vb shows a similar performance as Va, though Vb should in theory be slightly more expensive as the master needs to merge the feedback of both filtering steps. However, as communication dominates here and the timings are close to the measurement error, this is hardly visible.

Conclusions In general, the *broadcast/filter* strategy outperforms the *pre-filter/post-filter* strategy. The combination of a very large n and a rather small p marks one exception. The other exception is a memory critical setup: a mesh of size $n = 3584^2$, for example, does not fit into the memory for *broadcast/filter* as every rank on the master's compute node needs to store the complete mesh. The *pre-filter/post-filter* strategy postpones this limitation. Of course, an alternative is to pin the master rank to a separate compute node, but this deteriorates the usability. For the interested reader: the used Haswell nodes have a memory of 64GB while the flexibility-oriented implementation in preCICE roughly uses 200B per vertex. The most interesting physical scenarios feature typically a large n and a large p. Theses cases show a nearly uniform distribution of the runtime over all five steps of the *broadcast/filter* strategy. This means that the optimization of a single step has no crucial influence on the overall runtime. At the same time, step IIIa, for example, is already rather optimal. The only real performance boost could be realized via a multi-level re-partitioning scheme. This is a necessary step for an exa-scale ready preCICE, but not necessary for the current applications. Please recall, for ASTE, n and p are surface dimensions, not volume dimensions. Hence, $n = 1792^2$ and $p = 1792$ mark a huge testcase, for which a mesh re-partitioning time under 10s is fully tolerable. Section 4.5 compares these timings to realistic single-physics initialization times.

After the description of the mesh re-partitioning in this section, participant A holds decompositions of both meshes, Γ_A (user-defined) and Γ_B (re-partitioned), while participant B still holds its user-defined decomposition. Just next, Section 4.2 studies how data is communicated between both participants on Γ_B. Afterwards, Section 4.3 describes the realization of interpolation methods between both meshes at participant A and Section 4.4 the coupling schemes on Γ_B at participant B.

4.2 Communication on Distributed Data

An important ingredient for the parallel peer-to-peer concept is the local communication. It is challenging, since both participants possess, in general, different domain decompositions, which then also entail different decompositions of the coupling interface. Thus, for every rank, we need to identify which part of the coupling mesh it has to communicate to whom. Furthermore, a technical realization of these communication channels is necessary. This is also non-standard, since both participants are typically started in different `MPI_COM_WORLDS`. From a software engineering point of view, a reuse of the already existing 1:N communication in preCICE is desirable. It has proven to be very robust and it also ensures backward compatibility. The latter means, that the old server-based parallelization concept is still fully functional, which allows to port the various features one by one to the distributed version. This guarantees a smooth transition for the users and enables early feedback on new implementations for the developers.

The main requirement for the new local M:N communication is robustness. Deadlocks should never appear independent of the underlying implementations. Both kernel implementations, MPI and TCP/IP, should still be supported, the technical level should, however, be well-separated from the algorithmic level. Finally, the implementation should be fully local, hence ready for exa-scale, during each timestep. The initialization, however, can still feature global operations as long as they remain tolerable, compare the discussion in Section 1.3.2. Section 4.2.1 gives an algorithmic and technical description of the M:N communication. Afterwards, Section 4.2.2 briefly summarizes the implementation. Finally, Section 4.2.3 gives detailed performance results for the ASTE testcase.

4.2.1 Algorithmic and Technical Description

The starting point for the algorithmic consideration are two domain decompositions of the same mesh, one at participant A and one at participant B. To remain with the notation from the last section, these are

$$\Gamma_B = \bigcup_{i=0}^{p_B-1} \Gamma_B^i = \bigcup_{i=0}^{p_A-1} \tilde{\Gamma}_B^i \,.$$

Each domain decomposition can stem either from a local definition, such as for participant B, or from a re-partitioning, such as for participant A. It is sufficient to only consider vertices, since all data values are solely stored on them. Theses decompositions are now used to identify which rank of A has to communicate what to which rank of B and vice versa. The algorithmic initialization is described further down. I first focus on the technical realization.

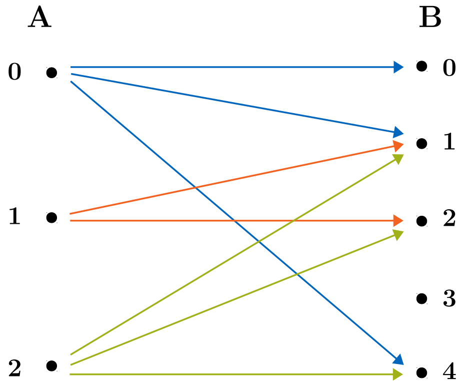


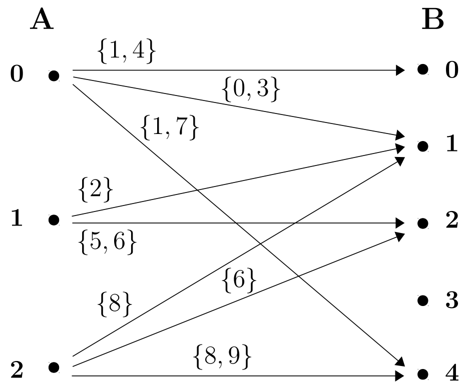
Figure 60: Example of an M:N communication between participant A (on three processors) and participant *B* (on five processors). The communication layout consists of three 1:N communications, distinguished by different colors. A similar figure is used in [43].

Technical Initialization Without loss of generality, *A* and *B* play the role of the acceptor and the requestor of the communication, respectively. This choice is completely independent of the role from the mesh re-partitioning in the last section. The roles are consistent with the classical nomenclature of, for example, MPI ports: the acceptor publishes connection information while the requestor pulls for such information. Technically, the M:N communication between *A* and *B* builds upon multiple 1:N communications, which mark the kernel of the communication. The 1:N kernel itself is either implemented via TCP/IP or via MPI-2.0 ports. I mention this already in Section 2.1.2. Also, [99] and [181] give detailed technical information. The acceptor of a kernel communication runs on one thread in a server-like way, while the requestor runs on N threads in a client-like way. To publish connection information of a single kernel communication, the acceptor writes to a hidden file. The requestor threads all read the connection file and establish the connection. Afterwards, the acceptor removes the file again. If multiple such kernel communications need to exchange connection information at the same time, creating a subfolder per connection speeds-up the file access significantly. Still, publishing connection information via the file system is not optimal. Section 4.2.3 gives detailed information on the performance. Figure 60 sketches exemplarily, how the global M:N communication is build upon M local 1:N communications. Every rank of *A* accepts one 1:N communication as a server, where N refers to the amount of ranks of B that it has to communicate with. This avoids deadlocks at initialization in a rather elegant way. Consider, for example, rank 1 of participant B in Figure 60: three 1:N communications need to be requested. This happens sequentially and in an arbitrary order. Since every rank of A only has to publish one connection information, all information is always available and no deadlock can occur. It is easy to see, that a 1:1 kernel communication would not allow for such a simple solution.

Technical Communication Please consider again Figure 60: for the actual communication, different messages need to be sent along every arrow, so to speak in a 1:1 way. Since the order of the channels at each rank is arbitrary, another measure is needed to further avoid deadlocks then. Asynchronous communication for all kernel implementations is a remedy. Possible alternatives are to start each send and receive operation in a single thread, possibly via OpenMP. This, however, has not proven to be reliable over a broad range of MPI implementations due to inconsistencies of the thread-safety guarantees [181]. Another idea would be a pulling mechanism, where the receiver opens one single `any-tag` communication channel for everybody and every message is tagged by the sender. This appears, however, rather tedious. We[42] decided to choose the asynchronous solution. It has the only drawback that additional developments are necessary since the original kernel communication in preCICE is blocking [99]. Besides the additional work, the solution is reliable, elegant and efficient [181].

Algorithmic Initialization I now review the initialization from an algorithmic perspective. Both decompositions, which are constructed either in the gather step I or in the feedback step V of the

[42] Alexander Shukaev an myself



	$\Phi_B(0) = \{1, 4\}$
$\Phi_A(0) = \{0, 1, 3, 4, 7\}$	$\Phi_B(1) = \{0, 2, 3, 8\}$
$\Phi_A(1) = \{2, 5, 6\}$	$\Phi_B(2) = \{5, 6\}$
$\Phi_A(2) = \{6, 8, 9\}$	$\Phi_B(3) = \{\}$
	$\Phi_B(4) = \{1, 7, 8, 9\}$
$\Psi_A^0(0) = \{1, 3\}$	$\Psi_B^0(0) = \{0, 1\}$
$\Psi_A^0(1) = \{0, 2\}$	$\Psi_B^1(0) = \{0, 2\}$
$\Psi_A^0(4) = \{1, 4\}$	$\Psi_B^1(1) = \{1\}$
$\Psi_A^1(1) = \{0\}$	$\Psi_B^1(2) = \{3\}$
$\Psi_A^1(2) = \{1, 2\}$	$\Psi_B^2(1) = \{0, 1\}$
$\Psi_A^2(1) = \{1\}$	$\Psi_B^2(2) = \{1\}$
$\Psi_A^2(2) = \{0\}$	$\Psi_B^4(0) = \{0, 1\}$
$\Psi_A^2(4) = \{1, 2\}$	$\Psi_B^4(2) = \{2, 3\}$

Figure 61: Example for the transformation from vertex distributions Φ_A and Φ_B to local communication maps $\Psi_A^{\tilde{p}_A}$, $\tilde{p}_A = 0, \ldots, 2$ and $\Psi_B^{\tilde{p}_B}$, $\tilde{p}_B = 0, \ldots, 4$. Left: sketch of communication channels and global vertex indices. Right: corresponding vertex distributions and local communication maps. Empty communication maps are suppressed for the sake of readability.

mesh-repartitioning strategies, compare Figure 55, are each stored as a so-called vertex distribution

$$\Phi : \{0, \ldots, p-1\} \to \mathbb{P}(\{1, \ldots, n\}) \ ,$$

where $\mathbb{P}$ denotes the power set. All vertices are identified by an integer ID. The IDs for a single rank $\Phi(\tilde{p})$ are stored in a vector, ordered naturally. In general, the decomposition for re-partitioned meshes is overlapping, compare Figure 54. At the start of the initialization, both master ranks only hold the vertex distribution of their decomposition. Thus, as a first step, the two vertex distributions are exchanged and both vertex distributions are broadcast to every rank afterwards. The broadcast is to some extent redundant as local information from the mesh re-partitioning could be reused. Still, from a software engineering point of view, I choose to fully decouple the M:N communication from the mesh-repartitioning. Thereby, both steps can be implemented and tested separately. Substituting any initialization step by an optimized variant at a later point in time should be easy. Now, every rank can locally extract which vertices it has communicate to whom. This information is stored in a so-called local communication map in A

$$\Psi_A^{\tilde{p}} : \{0, \ldots, p_B - 1\} \to \mathbb{P}(\{1, \ldots, n\}) \ \forall \, \tilde{p} = 0, \ldots, p_A - 1 \ ,$$

and similarly in B. Here, $\Psi_A^{\tilde{p}_A}(\tilde{p}_B)$ are the local vertex indices at A at rank $\tilde{p}_A$ that have to communicated to rank $\tilde{p}_B$ at B. $\Psi_A^{\tilde{p}_A}$ can be identified in a straight-forward way by iterating over $\Phi_A(\tilde{p})$ in an outer loop and over $\Phi_B(0), \ldots, \Phi_B(p_B - 1)$ in an inner loop. A complexity of $O(n^2/p)$ results if $\Phi_A(\tilde{p})$ holds $O(n/p)$ vertices. As this concept is not overly sophisticated and can be easily understood from the example in Figure 61, the fast reader might directly go to the next section.

4.2.2 Implementation

Figure 62 sketches the software architecture of the `com` package and the newly developed `m2n` package. For a general overview of all packages, please recall Table 1 and Figure 20. The main feature of the applied architecture is the clear separation of the technical implementation of the 1:N kernel communications in the `com` package from the logical layer in the `m2n` package. Thus, arbitrary combinations of both are possible at runtime. The `M2N` class manages the overall communication between two participants. For every coupling mesh that is used for communication, a `DistributedCommunication` is initiated. The `PointToPointCommunication` follows the description of the last section. In addition, a `GatherScatterCommunication` is implemented, which re-directs all communication through both master ranks. This variant is merely intended to validate the `PointToPointCommunication` during the development process. Both implementations of `DistributedCommunication` are created by associated factories to facilitate the configuration. Similarly, in the `com` package, further factories are added because the amount of kernel communications is unknown at configuration time. `MPICommunication` is an abstraction layer for both supported MPI variants: `MPIDirectCommunication`, which enables a start-up of both participants in the same MPI communicator, and `MPIPortsCommunication`, which enables the start-up in separated MPI communicators. `SocketCommunication` implements the TCP/IP communication.

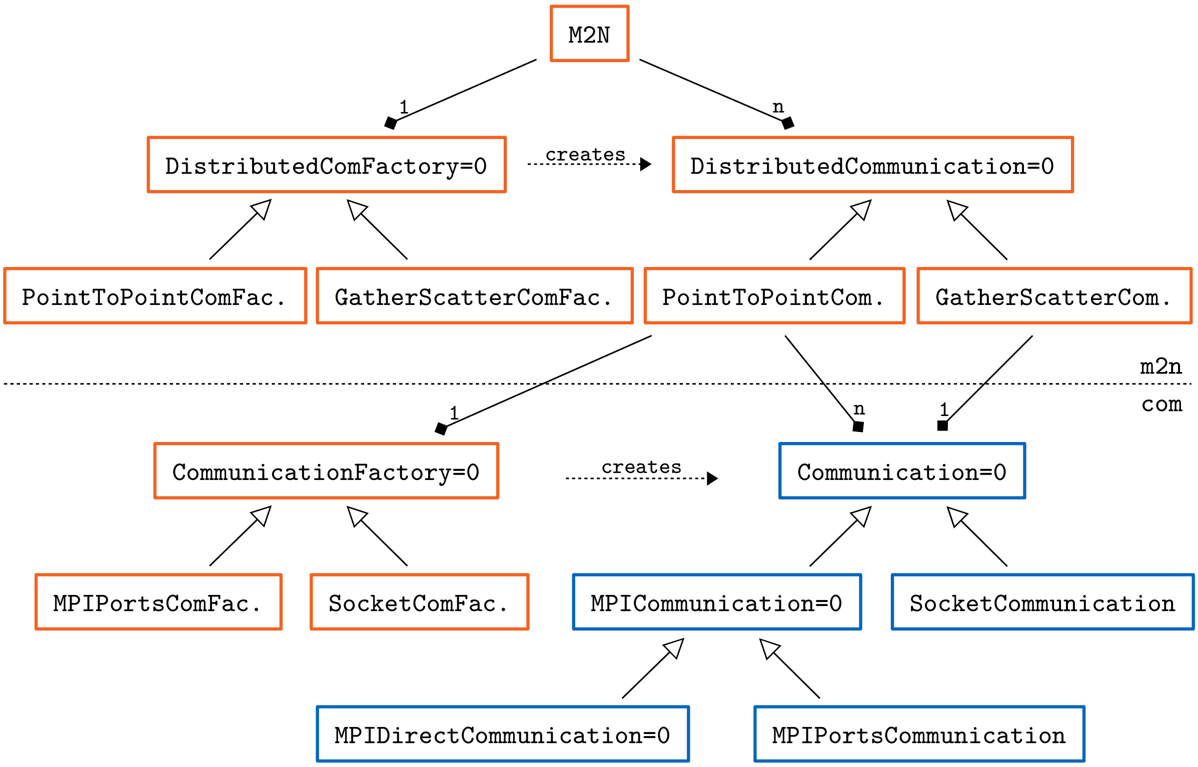


Figure 62: Software architecture of the `m2n` package. The arrow notation follows the UML standard (empty triangles: inheritence, filled diamonds: containement). Distributed communication logic in the `m2n` package is well separated from technical 1:N communication details in the `com` package. This allows for modular combinations of both components. Original components [99] are marked in blue, whereas new components are marked in orange.

4.2.3 Scaling Results

I, first, have a look at the initialization timings, before looking at the actual communication performance. If not stated differently, the master-slave communication is implemented via MPI, while the M:N communication uses the TCP/IP implementation. Later, I also discuss performance differences between both.

Building the Communication Maps To setup the M:N communication, the vertex distributions Φ of both participants need to be transformed into the local communication maps Ψ, compare Section 4.2.1. Figure 63 visualizes a strong scaling of the necessary steps for $n = 448^2$, for $n = 896^2$ and for $n = 1792^2$. The number of cores per participant is doubled four times from $p = 112$ to $p = 1792$. I solely show timings for the acceptor A. The requestor B features almost identical operations and shows no significant performance difference. The first building block is the communication of both vector distributions (CVD) between both master ranks. The runtime increases slightly with p, since the memory layout gets less and less compact. At the same time, the increase with n is only moderate, probably due to the same effect. Next, both vector distributions are broadcast (BVD) among all ranks, which shows a similar behavior as the communication. Please note that both steps are always below 0.1s and, thus, significantly cheaper than similar operations for the mesh re-partitioning, compare Section 4.1.3. This is due to the lower memory requirement of the vertex distributions compared to the

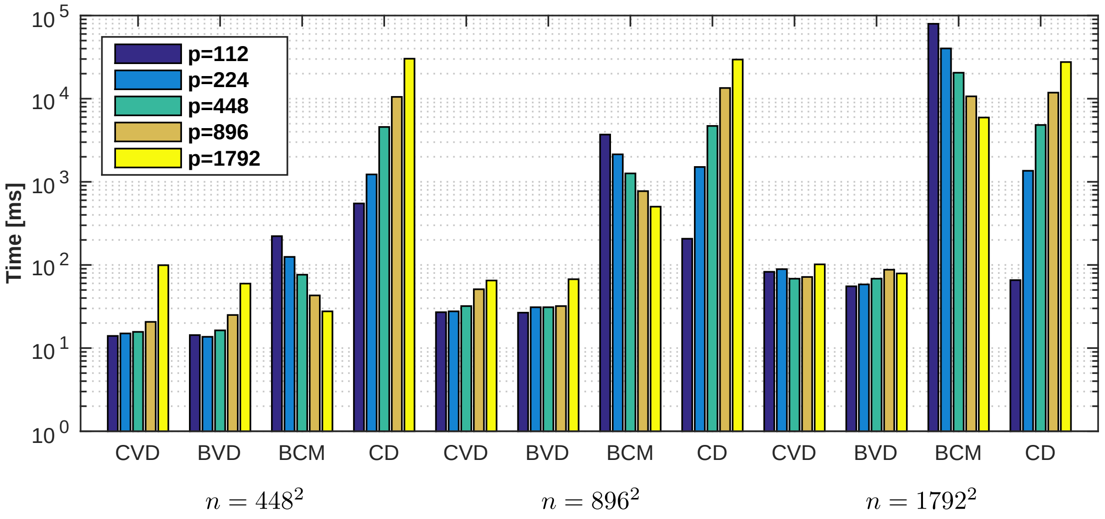


Figure 63: Building up communication maps: strong scalability results for three different mesh sizes. CVD: communicate vertex distributions Φ_A and Φ_B between the master ranks of both participants, BVD: broadcast vertex distributions Φ_A and Φ_B from the master rank to all slave ranks, BCM: build (local) communication map $\Psi_A^{p_A}$, CD: create directories. All timings are measured on the acceptor side A. The master-slave communication uses the MPI implementation whereas the inter participant communication is implemented by TCP/IP sockets. Please note that the CD steps includes a file system access, which induces fluctuations in the measurements.

associated mesh. Therefore, both steps are always almost negligible. In particular, not using the local re-partition information, but broadcasting the vertex distribution again, is fully justifiable. The third step is a pure compute step: the building of the local communication maps (BCM). As noted above, this step has a complexity of $O(n^2/p)$, which is clearly visible in the results. The quadratic dependence on n can be problematic for huge meshes, but it is still below 5s for $n = 1792^2$ and $p = 1792$, for example.

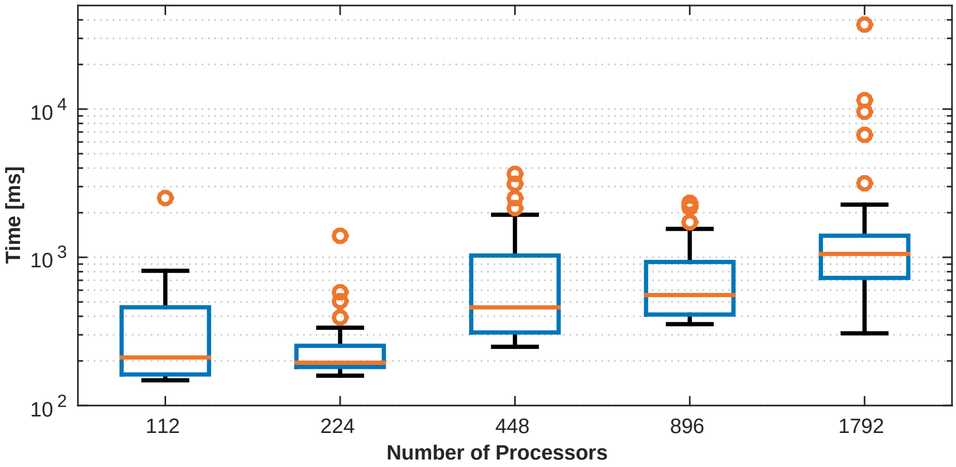


Figure 64: Communication establishment for TCP/IP. Connection information is published via the filesystem, which leads to an uncertain total establishment time. For each core count p, 30 measurements are listed. The orange line marks the median, while the blue box limits the second and third quartile. The whiskers enclose 1.5 times the interquartile range. Further outliers are marked explicitely. Please note the logarithmic scale.

Establishment of the Kernel Communications Figure 63 also lists the creation of directories (CD). This is a pre-step to the actual establishing of the 1:N communication channels and speeds-up the file access later. This step is independent of n and shows, despite the file system access, a relatively low variance. However, the runtime grows faster than linearly in p, though sequentially created by the acceptor's master rank. For $p = 896$, the runtime is still below 10s, but for a higher number of cores, this step becomes problematic. The establishment of the TCP/IP kernel communication itself shows a high runtime variance, since it features writing to and reading from the filesystem. Figure 64 shows a box plot with 30 measurements per core count p. The median rises as expected with p. Still, most measurements stay below 1s and are, thus, tolerable. The amount and runtime of outliers increases, however, significantly with p. Please note that Figure 64 uses a logarithmic scale. If the MPI implementation is used as underlying kernel communication, a completely different picture shows up. First, the Intel MPI implementation does not fully follow the standard[43]: if more than 8 compute nodes are used, this means more than $8 \cdot 28 = 224$ connections, the establishment of port connections between separate static `MPI_COM_WORLDS` fails. This was confirmed by Intel through the SuperMUC support. Increasing the environment variable `I_MPI_DAPL_UD_DIRECT_COPY_THRESHOLD` stepwise with the number of connections, can serve as a workaround here. Messages below this threshold use the DAPL UD direct-copy protocol. I could not measure a negative performance influence of this workaround though indicated in the Intel documentation[44]. For all Intel MPI tests, I use a threshold of 256kB. As this workaround makes the Intel MPI somehow usable, the IBM MPI implementation does not at all support port connections between separate static `MPI_COM_WORLDS`. This, again, was confirmed by IBM through the SuperMUC support. Furthermore, the Intel MPI routines itself show a significant overhead, which increases at least quadratically with p, up to 80s for $p = 1792$. Please note, that this overhead is completely due to the actual MPI implementation and therefore out of scope for a solution in preCICE.

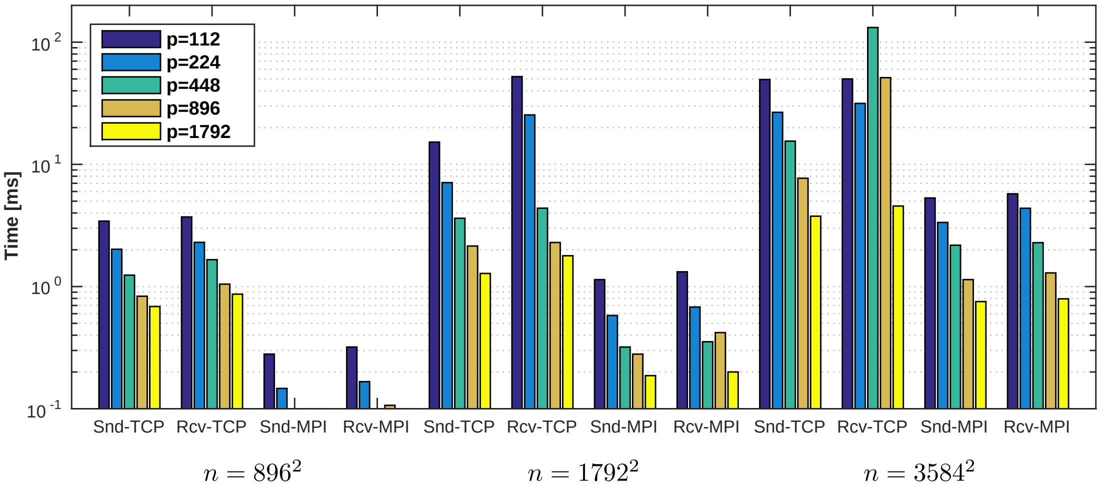


Figure 65: Data communication: strong scalability results for three different mesh sizes. All timings are measured on acceptor side A.

Communication of Data I first discuss the difference between the kernel implementations, MPI and TCP/IP. [99] reports a nearly similar performance for the original preCICE version. A case with $8 \cdot 10^6$ doubles, for example, needed 60.5ms with TCP/IP and 46.3ms with MPI on a QDR Infiniband for sending data back and forth once between two serial participants. The ASTE testcase needs 30.6ms with TCP/IP and 2.7ms with MPI on a FDR14 Infiniband. I use $n = 896^2$, which translates to $\approx 8.03 \cdot 10^6$ doubles, and also measure the time to send data back and forth once between two serial participants. The improvement of the TCP/IP communication relates very well to the difference between both Infinibands. The significant improvement for the MPI communication is, however, due to the asynchronous communication, which is newly implemented in [181]. The TCP/IP communication has always been asynchronous from a logical point-of-view. Figure 65 shows the actual communication time

[43] `http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report/node214.htm` and `.../node215.htm`
[44] `https://software.intel.com/en-us/node/528824`

for the strong scaling series $n = 896^2$, $n = 1792^2$, and $n = 3584^n$, for both variants. With the current implementations, MPI outperforms TCP/IP by a factor of 5 to 10 throughout all experiments. Figure 65 further shows the expected perfect speed-up with p and the expected linear scaling with n. Sending is typically faster than receiving, since the asynchronous send returns as soon as the buffer memory can be reused. This does, however, not mean that the message has been received. For $n = 3584^2$, an increase in communication time from $p = 224$ to $p = 448$ is visible. I assume that this is caused by a higher amount of smaller, but not yet small enough, messages. Please note that the fluctuations of the smaller runtimes are close to the measurement error of 1ms.

Conclusions I briefly summarize the main findings on the distributed communication. Beforehand, please note again that n and p relate to the mesh size and the number of cores at the coupling interface for an actual physical simulation. The communication and the compute effort in the initialization are fully tolerable. The latter might only be problematic for a high n and a small p. The current publishing strategy using the file system is tolerable until $p = 896$, but problematic afterwards. Alternatives, such as the internal MPI publishing routine, should be easy to integrate, though. The actual connection time differs significantly between both kernel implementations, TCP/IP and MPI. TCP/IP shows a robust and fast behavior. MPI is less reliable and can also be quite costly. The communication itself is very efficient for both variants, since it scales perfectly with p. MPI outperforms TCP/IP significantly, but both variants are still very fast. For a tremendous interface mesh of $n = 3584^2 \approx 1.3 \cdot 10^7$ and $p = 1792$, for example, a single communication step sending all surface mesh data from solver B to solver A is still below 5ms for both variants.

4.3 Interpolation Methods on Distributed Data

Interpolation between non-matching meshes is an important ingredient for any multi-physics simulation. If interpolation is supported, the discretization in every single-physics domain can be tuned for specific needs. For FSI simulations, for example, the fluid domain typically needs a much finer mesh close to the coupling interface to resolve boundary layers than the structural domain. At the same time, interpolation methods need to be chosen carefully, to conserve the right properties. Section 2.1.2 gives a compact literature review on interpolation methods. preCICE supports two different kinds of interpolation methods: projection-based mapping and radial-basis function (RBF) mapping. Both types are already supported in the serial preCICE version [99]. This section focuses merely on their realization on distributed data. The serial version of preCICE supported four different combinations of mapping variants: read-consistent, read-conservative, write-consistent, and write-conservative. The realization of the interpolation methods on distributed data simplifies drastically if only two out of these four variants are considered, namely read-consistent and write-conservative. To understand this restriction, Section 4.3.1 introduces basis mapping concepts and explains how projection-based mappings work, in serial as well as in parallel. Furthermore, I argue why this restriction has almost no practical consequences. Section 4.3.2 then focuses solely on the RBF mapping. The section encompasses an algorithmic description as well as technical parallelization details. As the runtime of the projection-based mapping schemes is almost negligible and furthermore already covered as part of the filtering in Section 4.1.3, Section 4.3.3 focuses on performance results for the RBF mapping solely.

4.3.1 Basic Concepts and Projection-Based Mapping

I start with basic notation. The coupling meshes are nothing else than clouds of vertices,

$$\Gamma_A = \{x_1^A, \ldots, x_{n_A}^A\} \text{ with } x_i^A \in \mathbb{R}^3 \ ,$$

and similarly for participant B. I restrict all mapping descriptions to the general three-dimensional case. The final goal of an interpolation method is to map scalar data $v_i^A \in \mathbb{R},\ i = 1 \ldots n_A$ associated with Γ_A onto $v_i^B \in \mathbb{R},\ i = 1 \ldots n_B$ associated with Γ_B. To simplify the further notation, these values are collected in vectors $V_A \in \mathbb{R}^{n_A}$ and $V_B \in \mathbb{R}^{n_B}$. If vectorial data, such as forces, needs to be mapped, each component is considered separately. I assume that the meshes Γ_A and Γ_B remain constant throughout the simulation. Thus, the same holds for the mapping. Note that this is true for an arbitrary-Lagrangian-Eulerian flow solver coupled to a Lagrangian structural solver, since the positions at the interface stay constant for both solvers. Only a pure Eulerian flow solver would contradict this assumption.

A remark on the notation: I use x_i here as the vertex coordinates. In Section 4.4 and Chapter 3, x_i refers to values on the vertices, so to speak to V_i here. This corresponds to the standard notation in the respective literature. The strict separation in different sections, should prevent the reader from any confusion.

Four Mapping Variants An interpolation method is nothing but a linear mapping and can, thus, be written as

$$V_A = H_{AB} V_B \ ,$$

with $H_{AB} \in \mathbb{R}^{n_A \times n_B}$. It is called consistent if the entries of every row sum up to one [31],

$$\sum_{j=1}^{n_B} (H_{AB})_{ij} = 1 \ \ \forall i = 1 \dots n_A \ .$$

This ensures the exact mapping of constant functions, which is typically a desired property for the mapping of deformations, fluxes or densities, for example. If the entries of every column sum up to one, a mapping is called conservative,

$$\sum_{i=1}^{n_A} (H_{AB})_{ij} = 1 \ \ \forall j = 1 \dots n_B \ .$$

This property, on the other hand, guarantees the conservation of the sum of the mapped values and is normally applied to integral values such as forces. Every consistent mapping H_{AB} induces a conservative mapping via $H_{BA} = H_{AB}^T$. preCICE further distinguishes between read and write mappings. To simplify this terminology, let us reconsider the standard setting, already used in Sections 4.1 and 4.2: Γ_B is sent from B to A and re-partitioned at A. This means that data is communicated via Γ_B and mapped from and to Γ_A at participant A. A read mapping is now applied before data is read from Γ_A, this means the mapping $\Gamma_B \rightarrow \Gamma_A$ is a read mapping. On the other hand, a write mapping is applied after data is written to ΓA. Hence, the mapping $\Gamma_A \rightarrow \Gamma_B$ is a write mapping. The Cartesian product of both categories finally gives the four mapping variants: read-consistent, read-conservative, write-consistent, and write-conservative. The original serial implementation in preCICE did not need to distinguish between read and write mappings in the computation of the mappings. This categorization was only needed to decide when a mapping is applied. Please note that transposing a mapping $H_{BA} = H_{AB}^T$ does not only transform a consistent mapping into a conservative one and vice versa, but also transforms a read mapping $\Gamma_B \rightarrow \Gamma_A$ into a write mapping $\Gamma_A \rightarrow \Gamma_B$ and vice versa, by inverting the mapping direction. Thus, the four variants are essentially two pairs: read-consistent and write-conservative, and read-conservative and write-consistent.

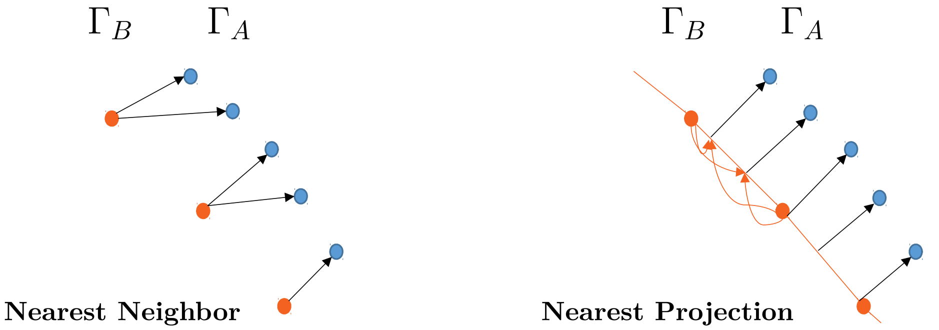


Figure 66: Schematic view of both projection-based mappings in a two-dimensional case. Both mappings are depicted in their consistent variant. Arrows point in data transfer direction. A similar figure is also used in [43].

Projection-Based Mappings Two kinds of projection-based mappings are supported in preCICE: the nearest neighbor mapping and the nearest projection mapping. To explain both, let us consider a consistent mapping $\Gamma_B \to \Gamma_A$. As explained further above, this comes without any loss of generality, as through transposing every consistent mapping induces a conservative mapping and vice versa. For the nearest neighbor mapping, every vertex of x_i^A of Γ_A looks for the closest vertex x_j^B of Γ_B and copies the value from x_j^B to x_i^A. Here, closest relates to the Eucledian distance. This method is only first order accurate (see e.g. [99]). Still, it plays an important role for many applications such as the handling of matching meshes with non-matching decompositions. The nearest-projection mapping needs additional connectivity information of Γ_B, meaning surface elements. In 3D, this can be triangle or quad elements, in 2D simple lines. Now, each vertex x_i^A of Γ_A searches for the closest element e_j^B of Γ_B and projects its coordinates orthogonally onto e_j^B. The values of the element's vertices are interpolated (bi-)linearly in the projection point. This value is then copied to x_i^A. If the orthogonal distance between x_i^A and e_j^B is much smaller than the element's width, this is a second order method (see e.g. [99]). This holds for most applications. Figure 66 sketches both projection-based mappings.

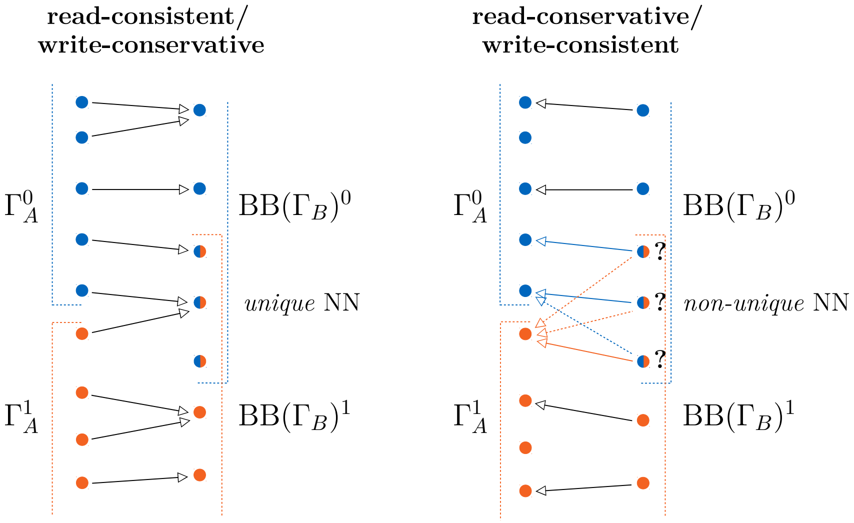


Figure 67: Schematic comparison between a read-consistent (resp. write-conservative) and a read-conservative (resp. write-consistent) nearest-neighbor mapping. The first combination leads to a unique mapping while the second one does not in the overlap region. $\mathrm{BB}(\Gamma_B)^0$ refers to mesh Γ_B, bounding box-filtered with respect to Γ_A^0. Γ_A is a local mesh, whereas Γ_B is received and re-partitioned. Arrows point in search direction.

Parallelization of Projection-based Mappings As mentioned above, I restrict the mapping variants from four to two to simplify the parallelization. These two valid variants are read-consistent and write-conservative, which are essentially identical, as explained above. Please recall the mesh repartitioning strategies from Section 4.1.1, Figure 55. In step IV, a mapping-dependent filter is applied. In Figure 67, this step is visualized for a nearest neighbor mapping. Γ_B is already filtered via a bounding box corresponding to each partition of Γ_A, denoted as $\mathrm{BB}(\Gamma_B)^i$. This results in an overlap. Next, preliminary mappings are computed. For the two supported variants, this is depicted on the left of Figure 67. Each vertex of Γ_A^i searches for the closest vertex in $\mathrm{BB}(\Gamma_B)^i$, visualized by arrows. Any vertex of $\mathrm{BB}(\Gamma_B)^i$ that is not found by this search, meaning that this vertex is not closest to any vertex of Γ_A^i, has no influence on the actual mapping and can be further filtered out. Since there is no overlap in the decomposition of Γ_A, this results in a unique nearest neighbor mapping. For the read-consistent mapping, when communicating the values on Γ_B from B to A, each value is simply sent to every duplicated vertex of Γ_B at A. Conversely, for the write-conservative mapping, when communicating the values on Γ_B from A to B, the values of the duplicated vertices are summed up. Thus, once the mesh-repartitioning is done, the mapping operates fully locally. The parallelization of the nearest neighbor mapping is somehow trivial then. The nearest projection mapping works similarly. Vertices can simply be filtered out if they do not belong to any nearest element. If two mappings of this valid group are combined, only vertices can be filtered out if they do not influence either of the two mappings.

Let us consider the invalid variants read-conservative and write-consistent, depicted on the right of

Figure 67. After the bounding box filtering, there are duplicated vertices of Γ_B, which now look for nearest neighbors in different local partitions of Γ_A. Each duplication finds a different nearest neighbor: the nearest neighbor mapping is not unique. This problem could be resolved by comparing the actual distance of every candidate. Further communication is necessary, which can be costly if not $n_B \ll n_A$. An alternative would be to generate an artificial overlap for the partitions of Γ_A. An exact and efficient solution seems tedious, though.

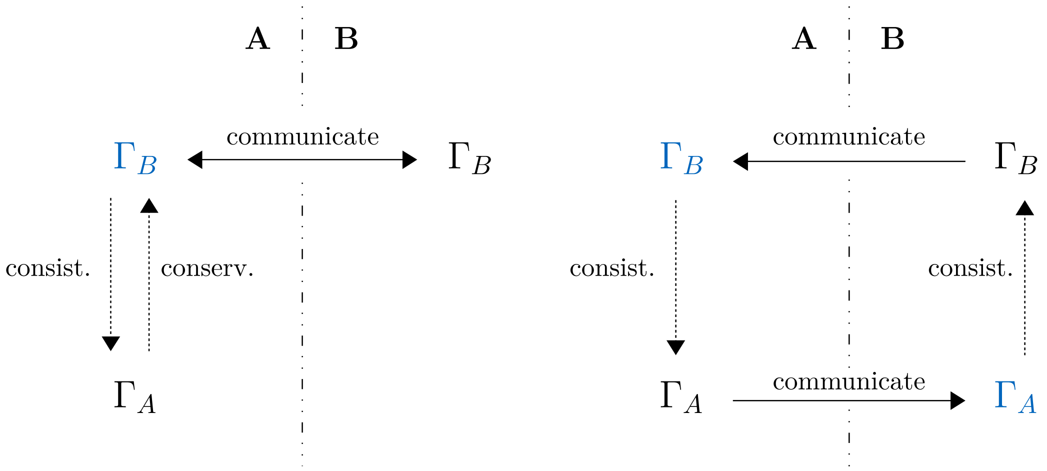


Figure 68: Valid mapping combinations. preCICE only allows consistent mappings from a re-partitioned mesh to a local mesh, i.e. in read direction, and conservative mappings in the reverse, write direction (left sketch). Changing such a conservative mapping to a consistent one requires the computation of the mapping on the other participant (right sketch). Now, both meshes need to be re-partitioned. Re-partitioned meshes are marked in blue.

I conclude to only support the first, valid mapping variants. For most applications, having these two variants is also fully sufficient. Typically, one of the two mappings between the solvers is required to be consistent whereas the other one has to be conservative. This is the case for many FSI simulations. Sometimes, this can be problematic as conservative mappings tend to induce spurious oscillations. [31, 220]. Then, consistent mapping in both directions is desirable. This can still be realized with only the two valid mapping variants, as a write mapping is transformed into a read mapping if it is moved to the other participant. This idea is depicted in Figure 68. The drawback is the necessary communication of data on both meshes. For very asymmetric cases, $n_B \ll n_A$ or $n_B \gg n_A$, the communication of only the smaller mesh would be favorable. Still, requiring both mappings to be consistent and having such asymmetric meshes at the same time is a rare case. Section 4.3.2 shows that the restriction to the first, valid mapping variants is also very useful for the RBF mappings.

As this section already discusses the algorithmic concepts and the parallelization of projection-based mappings, the next section solely focuses on the RBF mapping.

4.3.2 Radial Basis Function Mapping

I start with a brief mathematical description and focus afterwards on the parallelization. For a more detailed introduction to RBFs, the reader may refer, for example, to [31, 99]. As mentioned in the last section, it is sufficient to only consider the consistent variant from Γ_B to Γ_A. The general idea of the RBF mapping is to build up a global interpolation on Γ_B, which is then evaluated on Γ_A. This global interpolant is build up by radially symmetric basis functions centered at the vertices x_i of Γ_B,

$$\phi : \mathbb{R} \to \mathbb{R},\ \|x\|_2 \mapsto \phi(\|x\|_2),\ x \in \mathbb{R}^3\ .$$

In literature, various such basis functions are considered, either with local or global support, compare, e.g., [185]. Table 23 lists those that are supported in preCICE.

To ensure exact interpolation of constant and linear functions, the interpolant is further enriched by a

	Basis Function ϕ	Support
Gaussian	$\exp\left(-\left(a\|x\|_2\right)^2\right)$	global
Multiquadrics	$\sqrt{a^2+\|x\|_2^2}$	global
Inverse Multiquadrics	$1/\sqrt{a+\|x\|_2^2}$	global
Thin Plate Splines	$\|x\|_2^2\log\left(\|x\|_2\right)$	global
Volume Splines	$\|x\|_2$	global
Compact Thin Plate Splines C2	$1-30\xi^2-10\xi^3+45\xi^4-6\xi^5-60\xi^3\log\xi$	local
Compact Polynomial C0	$(1-\xi)^2$	local
Compact Polynomial C6	$(1-\xi)^8(32\xi^3+25\xi^2+8\xi+1)$	local

Table 23: Radial basis functions implemented in preCICE. a denotes a shape parameter. Local basis functions posses a support radius of r, i.e. $\phi(\|x\|_2)=0$ for $\|x\|_2>r$. ξ denotes normalized coordinates $\xi=\|x\|_2/r$. A similar table is used in [42, 43, 99].

global first order polynomial. The interpolant $s:\mathbb{R}^3\rightarrow\mathbb{R}$ then reads

$$s(x)=\sum_{i=1}^{n_B}\gamma_i\cdot\phi(\|x-x_i^B\|_2)+q(x)\ ,$$

with the global linear function $q(x)=\beta_0+\beta_1x_1+\beta_2x_2+\beta_3x_3$. The goal is to find $\gamma_i\in\mathbb{R}, i=1\dots n_B$ and $\beta_i\in\mathbb{R}, i=0\dots3$ that fulfill the interpolation condition

$$s(x_i^B)=v_i^B\ \ \forall i=1\dots n_B\ .$$

This underdetermined system is regularized by the polynomial condition

$$\sum_{i=1}^{n_B}\gamma_i\cdot p(x_i^B)=0\ ,$$

for every polynomial $p:\mathbb{R}^3\rightarrow\mathbb{R}$ of degree less or equal than q, hence for every constant and linear function, compare, e.g., [185]. In matrix notation, these conditions read

$$\underbrace{\left(\begin{array}{c|c} 0 & P_B^T \\ \hline P_B & M_{BB} \end{array}\right)}_{=:C}\left(\begin{array}{c} \beta \\ \hline \gamma \end{array}\right)=\left(\begin{array}{c} 0 \\ \hline V_B \end{array}\right)\ , \tag{6}$$

where $M_{BB}\in\mathbb{R}^{n_B\times n_B}$, $(M_{BB})_{i,j}=\phi\left(\|x_i^B-x_j^B\|_2\right)$ and the ith row of $P_B\in\mathbb{R}^{n_B\times4}$ looks like $(1\,x_{i,1}^B\,x_{i,2}^B\,x_{i,3}^B)$. Here, $x_{i,1}^B, x_{i,2}^B, x_{i,3}^B$ refer to the components of x_i^B. Furthermore, $\gamma=(\gamma_1,\dots,\gamma_{n_B})^T\in\mathbb{R}^{n_B}$ and $\beta=(\beta_0,\dots,\beta_3)^T\in\mathbb{R}^4$.

Afterwards, the interpolant can be evaluated at the vertices of Γ_A,

$$v_j^A=s(x_j^A)=\sum_{i=1}^{n_B}\gamma_i\phi(\|x_j^A-x_i^B\|_2)+q(x_j^A)\ \ \forall j=1\dots n_A\ ,$$

or again in matrix notation

$$\left(\begin{array}{c} V_A \end{array}\right)=\underbrace{\left(\begin{array}{c|c} P_A & M_{AB} \end{array}\right)}_{=:D}\left(\begin{array}{c} \beta \\ \hline \gamma \end{array}\right)\ , \tag{7}$$

with $M_{AB}\in\mathbb{R}^{n_A\times n_B}$, $(M_{AB})_{i,j}=\phi\left(\|x_i^A-x_j^B\|_2\right)$ and the ith row of $P_A\in\mathbb{R}^{n_A\times4}$ like $(1\,x_{i,1}^A\,x_{i,2}^A\,x_{i,3}^A)$.

For the consistent mapping now simply first solve (6) and then evaluate (7). As I assume that the mesh Γ_B does never change, neither does C. To map different values, we only need to adjust the right-hand-side of (6). Thus, a pre-computation of an LR-decomposition of C would pay off. The serial version of preCICE uses such a strategy. This, however, is not advisable for the realization on distributed data. Furthermore, if local basis functions are used, C becomes sparse and an LR-decomposition is overly expensive then. Still, a constant, possibly expensive, pre-conditioner should also take advantage of the constant C. Before I continue with the parallelization, let us have a brief look on the induced conservative variant. In compact form, the consistent mapping $\Gamma_B \to \Gamma_A$ reads

$$V_A = H_{BA} V_B = D C^{-1} \begin{pmatrix} 0 \\ V_B \end{pmatrix} .$$

Thus, the conservative variant is

$$V_B = H_{AB} V_A = H_{BA}^T V_A = C^{-1} D^T \begin{pmatrix} 0 \\ V_A \end{pmatrix} .$$

Hence, the input values are first multiplied with D and then, again, a C system is solved.

Parallelization Similar to the last section, for parallelization, I restrict to the read-consistent variant $\Gamma_B \to \Gamma_A$ with, as usual, Γ_B being the re-partition mesh and Γ_A the local mesh. I only consider RBFs with local support. RBF with global support do not allow to efficiently distribute data. Both matrices, C and D, are decomposed row-wise. For D, this comes naturally with the local, non-overlapping, partitions of Γ_A. For C, this is more involved. It is easy to see that the re-partitioning of Γ_B must result in overlapping partitions to be able to fill all entries of D. Therefore, we must decide upon one owner rank for duplicated vertices of Γ_B that actually holds the respective row of the matrix. Additionally, the master rank of participant A also holds the first four rows of C, which are the polynomial rows. For many single physics solvers this is a decent load balancing choice as the master rank itself holds no vertices. As C is a symmetric matrix, it is sufficient to only fill up the lower triangle. Thus, the master does not need to hold the complete mesh Γ_B to fill up the polynomial rows.

The mapping-specific filtering takes care of the correct decomposition as well as the assingment of ranks. Please recall the mesh-repartitiong, Section 4.1.1, Figure 55, step IV. For an RBF mapping, this step is slightly more involved than for projection-based mapping schemes. Figure 69 sketches the necessary three sub-steps. First, a bounding box around the local partition Γ_A^i is computed. This bounding box is extended in every direction by the support radius of the underlying RBF. Every vertex of Γ_B that lies within this bounding box is tagged as a possible candidate to be owned by the current rank (step I). All tagged vertices need to be part of the final local re-partition $\tilde{\Gamma}_B^i$ to ensure the correct construction of matrix D. Next, the master rank of B assigns an owner rank to every vertex. If multiple candidates for a single vertex are available, a first round of assignments tries to balance the load equally among all ranks, while a second round then assigns the remaining vertices in a greedy way (step II). Finally, a further bounding box is constructed around all owned vertices of Γ_B, again extended by the support radius. All vertices of Γ_B that lie outside this bounding and that were not tagged before, are now filtered out (step III). This second bounding box ensures the correct construction of matrix C.

The linear algebra library PETSc [6] is used to manage the decomposed matrices and to solve system (6). PETSc offers a very broad range of parallel linear solvers. For the sake of concision, I skip a description of PETSc here. Please refer to the PETSc webpage[45]. preCICE uses a sparse matrix format to store both matrices, though the polynomial rows of C are always dense. The RBF system (6) is solved via a GMRES solver [173], which is known to be very robust. PETSc uses a block-Jacobi preconditioner as default. This choice is problematic if the master rank holds no vertices, but solely the polynomial rows. A standard Jacobi preconditioner is the better choice then. As matrix C does not change during the simulation, the preconditioner needs only to be computed once. PETSc automatically takes care of this. preCICE starts each GMRES iteration from a zero vector. Restarting from a previous solution might be beneficial. After solving (6), PETSc is also used for the matrix multiplication (7). As stated above, for the write-conservative mapping, the matrix multiplication precedes the system solution.

A final brief remark on the excluded cases: a read-conservative (or a write-consistent) mapping would need an overlap region of the local mesh. All in all, this is very similar to the already discussed projection-based mappings. A realization of these variants, is not unfeasible, but far more tedious and of limited

[45] https://www.mcs.anl.gov/petsc/

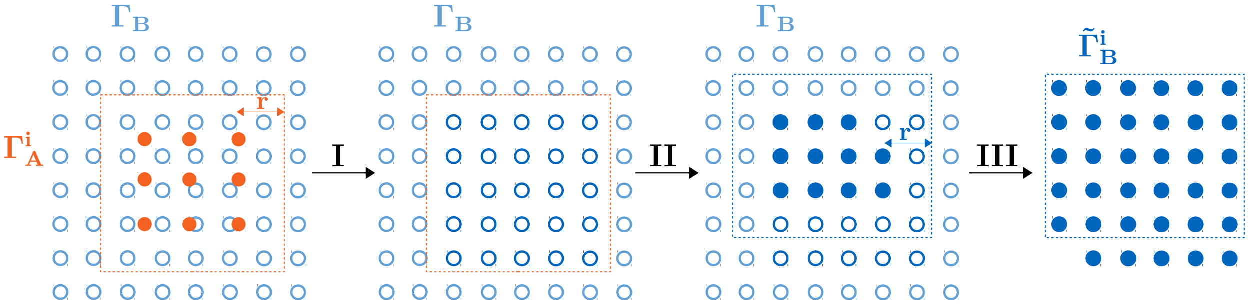


Figure 69: Schematic view on the RBF-based filtering. Step I: all vertices of Γ_B that lie within a bounding box around the local partition Γ_A^i plus the support radius r are tagged (dark blue, empty circles). This ensures the correct construction of matrix D. Step II: via a global procedure, a sub-group of the tagged vertices becomes owned vertices of rank i (dark blue, filled circles). Step III: Γ_B is filtered to the local re-partition $\tilde{\Gamma}_B^i$ via a bounding box around all owned vertices plus the support radius r. This further ensures the correct construction of matrix C. Previously tagged vertices are, however, not filtered out to not destroy the correct construction of Matrix D.

practical relevance. Furthermore, to also allow for RBF with global support, the re-partitioned Γ_B can simply not be filtered. Every rank needs to hold the complete mesh to be able to fill up his rows of C. This renders a local communication between both participants impossible.

4.3.3 Scaling Results

In this section, I solely focus on the performance of the RBF mapping. The projection-based mappings show a negligible compute effort, since, once the filtering is applied, their initialization has a complexity of $O(n^2/p^2)$. In [43], for example, the ASTE testcase with $n = 512^2$, $p = 1024$ on the thin nodes partition of SuperMUC phase 1 results in a compute time for the nearest-neighbor mapping below 1ms.

ASTE is a very challenging testcase for the RBF mapping, since the line-wise domain decomposition leads to very large overlap regions. Please recall the introduction of ASTE at the beginning of this chapter. For all experiments, I use compact thin plate splines as RBF with a support radius of $r = 2.5/\sqrt{n}$. Thus, two neighboring nodes in every direction are covered. In practice, this is a reasonable, yet minimal, choice. ASTE writes random data, uniformly distributed in $[0, 1]$, to the coupling meshes. I let PETSc iterate up to a relative convergence criterion of 10^{-5}. Furthermore, I only list results for the consistent mapping. The results for the conservative mapping do not show any significant difference. This comes at no surprise: as data values are assigned randomly, both variants are, in theory, nearly identical concerning the computational costs.

Figure 70 visualizes three strong scaling series for $n = 112^2$, $n = 224^2$, and $n = 448^2$. The number of cores doubles three times from $p = 28$ to $p = 224$. The initialization of the RBF mapping appears rather costly. It increases sub-linearly with p. Only from one node, $p = 28$, to two nodes $p = 56$, a speed-up is visible. Far more serious, though, is the increase in runtime with n. The increase appears super-linear, and quickly passes a tolerable amount. For example, for $n = 448^2$ and $p = 224$, the initialization takes already more than 4 minutes. The overall cost is not caused by the filling of the matrices, though. The time for filling both matrices C and D, shows a speed-up with p as expected. The costly part is the actual assembling of the matrices in PETSc – allocation of memory and construction of the sparse matrix format. To some extent, this is understandable as the assembling of the polynomials rows needs global communication. The work per timestep now looks much better. I only list the timings for solving (6), since all other building blocks, such as copying values back and forth between the preCICE data structures and the ones of PETSc, or the matrix multiplication in (7), are negligible. In Figure 70, I distinguish between the first mapping of data and all following as the first one includes the computation of the pre-conditioner. Both show a difference up to a factor of 10. The dependence on p is only minor. In particular, no speed-up is visible. This is probably due to the global polynomial rows. Positive is the only moderate dependence on n, which appears sub-linear, contrary to the initialization. For

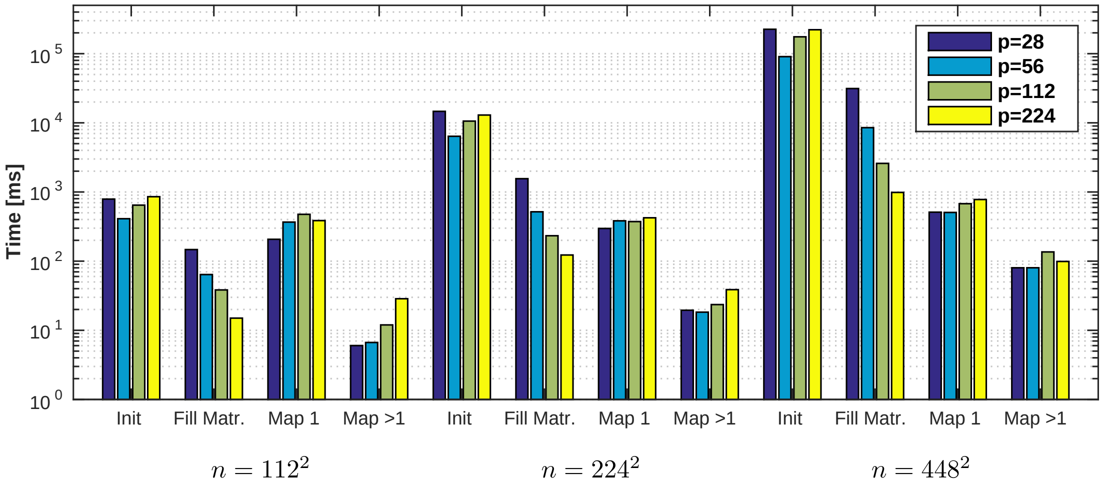


Figure 70: Scalability results for ASTE using a consistent RBF mapping. The first mapping is listed separately from all following, since the pre-conditioner is only computed once.

$n = 448^2$, for example, a single mapping is always below 0.1s. Please note that, due to the random data, fluctuations are natural.

Conclusions Contrary to the projection-based mappings, which show a negligible computational cost, RBF mappings have to be applied with caution. They feature a significant initialization time, which increases super-linearly with the number of vertices. Here, a more detailed look into PETSc to find a better assembling strategy should be worth the trouble. The actual solving benefits highly from the constant system matrix C. More expensive pre-conditioners might still pay off and result in a further speed-up. In principle, the global polynomial makes the classical RBF mapping not very well suited for massively parallel applications. Section 2.2 gives a literature review on current work that tries to overcome this drawback. Finally, please note again that ASTE features a rather artificial and non-optimal domain decomposition. Section 4.5.2 gives performance results for a real physical application including RBF mappings.

4.4 Coupling Schemes on Distributed Data

After discussing the parallelization of the communication in Section 4.2 and the parallelization of the interpolation methods in Section 4.3, the last missing block that needs to be ported to distributed data is the coupling schemes block. I explain in Chapter 3 that a coupling scheme is simply a combination of a fixed-point equation and a convergence acceleration method. In this section, I discuss how the acceleration methods can be realized on distributed data. Therefore, I first repeat the general setting from Chapter 3: given $H : \mathbb{R}^n \to \mathbb{R}^n$, we seek for $\hat{x} \in \mathbb{R}^n$ with $H(\hat{x}) = \hat{x}$ by an iterative procedure

$$x^k \mapsto \tilde{x}^k := H(x^k) \mapsto x^{k+1} \ .$$

Here, the second mapping is the acceleration. A simple acceleration scheme is the Aitken underrelaxation:

$$x^{k+1} = \omega^k \cdot \tilde{x}^k + (1 - \omega^k) \cdot x^k \ .$$

$$\omega^k = -\omega^{k-1} \frac{(R^{k-1})^T \left(R^k - R^{k-1}\right)}{\|R^k - R^{k-1}\|_2^2} \ ,$$

where $R^k := \tilde{x}^k - x^k$ is the current residual. $x \in \mathbb{R}^n$ lives on the vertices of the coupling mesh. Thus, a decomposition of x is induced naturally. x might consists of several coupling variables, i.e., several sub-vectors, but those are again already decomposed. Please note the slightly different role of n in this notation. It is still related to the number of vertices, but can be a multiple of it, depending on the dimensions of the coupling variables and the concrete fixed-point equation. If the mesh re-partitioning results in overlaps in the mesh, the duplicated vertices are simply accounted twice in the fixed-point equation. This, of course, can slightly influence the convergence speed. In the classical setting, however, the acceleration scheme is computed on the smaller, communicated mesh on the local side – in the standard setting on Γ_B on participant B, the side where no mapping is computed. Thus, the mesh is locally defined and has no overlaps.

The parallelization of simple schemes such as the Aitken underrelaxation is trivial: the inner products simply result in all-reduce steps. Similarly, the parallelization of the convergence measures is trivial. More complex is the parallelization of the multi-secant methods, Anderson acceleration and generalized Broyden, which is the topic of the remainder of this section. Therefore, in Section 4.4.1, I revisit these two multi-secant methods along with a more compute-oriented discussion of their numerical kernels. The main kernel is a QR-decomposition, which uses Given's rotations as an update rule in contrast to the pure Gram-Schmidt orthogonalization of the original implementation in preCICE [99]. Afterwards, Section 4.4.2 describes step-by-step how these kernels are realized on distributed data and analyzes their complexity. Finally, Section 4.4.3 gives scaling results for both multi-secant methods.

4.4.1 Numerical Kernels of Multi-Secant Methods

I start with a brief repetition of the basic concepts of Chapter 3. Anderson acceleration and generalized Broyden both approximate the inverse Jacobian J^{-1} of the residual operator by collecting input and output information from previous iterations. Input information is decoded by differences $\tilde{x}^i - \tilde{x}^{i-1}$ and is collected column-wise in $W \in \mathbb{R}^{n\times m}$. Output information relates to the differences $\tilde{R}^i - \tilde{R}^{i-1}$ stored columns-wise in $V \in \mathbb{R}^{n\times m}$. Here, m refers to the amount of columns. This notation suppresses details which timestep either column stems from or if certain columns are filtered out. Important for the computational cost is that, in general, $m \ll n$. The approximate inverse Jacobian should satisfy the multi-secant equation

$$J^{-1}V = W \ .$$

The two multi-secant methods differ in their second condition, which is needed to get a unique approximation. Anderson acceleration minimizes the Jacobian itself

$$\|J^{-1}\|_F \to \min \ ,$$

which leads to a simple least-squares problem. The generalized Broyden method tries to implicitly incorporate information from previous timesteps by

$$\|J^{-1} - J^{-1,(N)}\|_F \to \min \ ,$$

where the superscript (N) refers to the previous timestep. This results in an update formula for the inverse Jacobian.

For an overview on both methods, please recall Algorithm 1 in Section 3.2. The main building block of both methods is a QR-decomposition of V. To use a QR-decomposition is more stable and allows for a more efficient implementation than the alternative direct computation of the pseudo-inverse $(V^TV)^{-1}V^T$ [78, 217]. The QR-decomposition reads $V = \hat{Q} \cdot \hat{U}$ with the orthonormal matrix $\hat{Q} \in \mathbb{R}^{n\times n}$, meaning $\hat{Q}^T\hat{Q} = I$, and the upper triangular matrix $\hat{U} \in \mathbb{R}^{n\times m}$, meaning $\hat{U}_{ij} = 0$ for $i < j$. I use U instead of R to denote the upper triangular matrix to avoid any confusion with the residual operator. As $\hat{U}$ features a lower zero block, we can restrict the QR-decomposition to its so-called economical variant by only taking the first m rows of $\hat{U}$. Then, we also only need to keep the first m columns of $\hat{Q}$ and get $V = Q \cdot U$ with the restricted matrices $U \in \mathbb{R}^{m\times m}$ and $Q \in \mathbb{R}^{n\times m}$. From iteration to iteration, V only differs by single columns, which are either added on the left or deleted from the right. Therefore, an updating scheme of the QR-decomposition is more efficient than a full re-computation in every iteration as originally implemented in preCICE [99]. Furthermore, as I explain in the next section, such an update scheme is fairly easy to parallelize. Algorithm 3 lists both basic operations, *add column* and *delete column*. While the deletion of the right-most column leads to a trivial restriction of Q and U, adding a column on the left needs some further computational effort to re-establish a valid QR-decomposition. First, the new column v is orthogonalized against Q via a modified Gram-Schmidt procedure. Afterwards, a series of Given's rotations are used to eliminate the newly created sub-diagonal entries in U and update Q

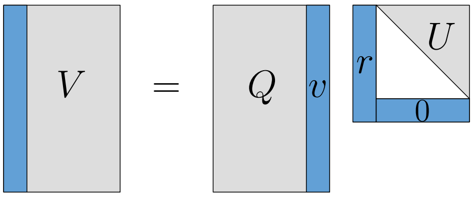
accordingly. The standard Given's rotation matrix reads

$$G(i,j,\theta) = \begin{array}{c}\begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix}\end{array} \begin{array}{c} \\ \\ j \\ \\ i \\ \\ \\ \end{array},$$

with $s = \sin(\theta)$ and $c = \cos(\theta)$. As the rotation $G(i,j,\theta)$ is an orthogonal transformation, it does not interfere with the orthogonality of Q. θ is adjusted such that a_{ij} rotates to zero for $G(i,j,\theta)A$, $A \in \mathbb{R}^{n\times n}$. A simple calculation shows that $\cos(\theta) = a_{ii}/\sqrt(a_{ii}^2 + a_{ij}^2)$ and $\sin(\theta) = -a_{ij}/\sqrt(a_{ii}^2 + a_{ij}^2)$. Thereby, rows i and j are possibly polluted meaning that non-zero entries are newly introduced. Thus, to bring U back to an upper triangular structure, first, the second till last entries of the first column are eliminated. Afterwards, the whole procedure is repeated for the submatrix $R(2:n,2:n)$. In total, we get $(m/2)\cdot(m-1)$ rotations. For details, in particular on a robust implementation, I want to refer the reader to [55]. In the following, I refer to a single *add column* step as **(QR)**. A final remark: if a general column is filtered out, compare Section 3.6.2 for the filtering algortihms of columns, also *delete column* requires a re-structuring of U. As this works fully analogue to the just described *add column*, I skip further details here.

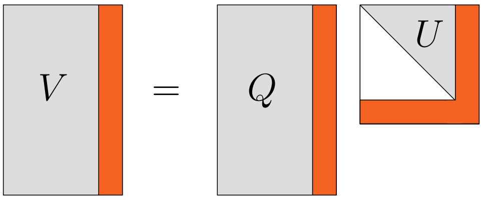
Algorithm 3 Updating the QR-decomposition of V. In contrast to [133], new columns are added on the left whereas only the rightmost column can be deleted. Sub-matrices are denoted in a MATLAB-like notation. A similar description is already used in [43].

Add Column
Input: $Q \in \mathbb{R}^{n\times m}$, $U \in \mathbb{R}^{m\times m}$, $v \in \mathbb{R}^n$
Output: $Q \in \mathbb{R}^{n\times(m+1)}$, $U \in \mathbb{R}^{(m+1)\times(m+1)}$



```
$r = 0^{(m+1)\times 1}$
for $j = 1 \dots m-1$ do
    $r(j) = Q(:,j)^T \cdot v$
    $v = v - r(j) \cdot Q(:,j)$
end for
$Q(:,m+1) = v/\|v\|$
$U = \left[ r, \begin{pmatrix} U \\ 0 \end{pmatrix} \right]$
compute Givens rotations s.t.
$U = G_{1,2}G_{2,3}\dots G_{m,m+1}U$ is upper triangle
$Q = QG_{m,m+1}\dots G_{1,2}$
```

Delete Column
Input: $Q \in \mathbb{R}^{n\times m}$, $U \in \mathbb{R}^{m\times m}$
Output: $Q \in \mathbb{R}^{n\times(m-1)}$, $U \in \mathbb{R}^{(m-1)\times(m-1)}$



```
$U = U(1:m-1, 1:m-1)$
$Q = Q(:, 1:m-1)$
```

Once the QR-decomposition of V is updated, both multi-secant methods differ in their next steps. The Anderson acceleration solves a backward substitution $U\alpha = -Q^T R^k$ for the coefficients $\alpha \in \mathbb{R}^m$ **(BS)**, followed by a linear combination of the input values $\Delta\tilde{x} = W\alpha$ **(WA)**. The generalized Broyden method, on the other hand, first solves m different backward substitutions $UZ = Q^T$ for $Z \in \mathbb{R}^{m\times n}$ **(MBS)**. Afterwards the inverse Jacobian is updated $J^{-1} = J^{-1,(N)} + (W - J^{-1,(N)}V)Z$, split in three steps: first $\widetilde{W} = (W - J^{-1,(N)}V)$ **(JV)**, second $\Delta J^{-1} = \widetilde{W}Z$ **(WZ)**, and third $J^{-1} = J^{-1,(N)} + \Delta J^{-1}$. Finally, the update $\Delta\tilde{x} = -J^{-1}R^k$ **(JR)** is computed. In the end, both multi-secant methods apply the update $x^{k+1} = \tilde{x}^k + \Delta\tilde{x}^k$.

4.4.2 Porting the Numerical Kernels to Distributed Data

After the introduction of all numerical kernels in the last section, this section now describes their realization on distributed data. I discuss them one by one with a detailed view on their complexity. The latter means the dependence of their computational cost on n, m, and p. I assume an ideal load balancing, where each rank holds n/p vertices. Please note that for real applications this is normally not the case as the domain decomposition of a single-physics solver is optimized towards its domain and not towards its coupling surface. Section 4.5.2 shows the load balancing of an actual physical FSI example.

QR Decomposition on Distributed Data (QR) The decomposition of the coupling meshes and hence x^k naturally induces a row-wise decomposition of the input/output matrices V and W and the matrix Q, as depicted in Figure 71. As U is of the limited size $m \times m$, a copy of the matrix is held at every rank. As Algorithm 3 shows, the first step of *add column* consists of a Gram-Schmidt orthogonalization of the new column v to all columns of Q. As this step simplifies to dot products, the parallelization simply uses all-reduce operations. A total of m dot-products of local length n/p are necessary. If we assume a logarithmic cost for the allreduce step, the complexity becomes $O(mn/p) + O(m \log p)$. Afterwards, the Given's rotations operate fully locally. Since every rank holds the same U, the same rotation is computed everywhere and is simply applied to the local part of Q. As mentioned in the last section, $m/2 \cdot (m-1)$ Given's rotations are applied, each one adds two columns of Q of length n/p and two rows of U of length m. Thus, a complexity of $O(m^2 n/p) + O(m^3)$ results. For the sake of completeness: *delete column* has an obvious constant complexity $O(1)$.

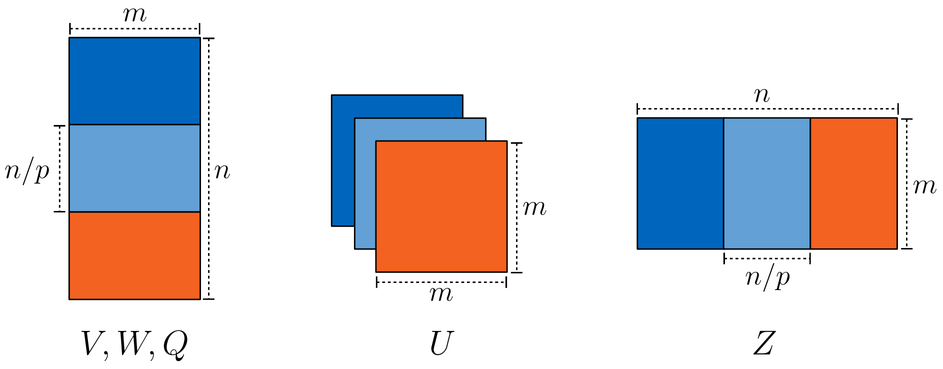


Figure 71: Decomposition of various matrices on distributed data. V, W and Q are composed row-wise and Z column-wise, while each processor holds a copy of U.

Backward Substitution on Distributed Data (BS / MBS) Anderson acceleration uses a backward substitution to resolve $U\alpha = -Q^T R^k$ for $\alpha \in \mathbb{R}^m$. Therefore, the right-hand side $-Q^T R^k$ first needs to be computed. As Q is decomposed row-wise, every rank computes its contribution locally at a cost of $O(mn/p)$. Afterwards, in an all-reduce step, the contributions are summed up, resulting in $O(m \log p)$. Next, each rank solves the identical triangular system, $O(m^2)$. In total, this results in $O(mn/p) + O(m \log p) + O(m^2)$. The generalized Broyden method performs this backward substitution n times – for every unit vector e_i, $i = 1 \dots n$ instead of R^k – to resolve $UZ = Q^T$. However, each rank can do its n/p part individually. This corresponds to a column-wise decomposition of Z, compare also Figure 71. Here, no communication is necessary. The total cost is $O(nm^2/p)$

Compute Update $\Delta\tilde{x} = W\alpha$ on Distributed Data (WA) The update step of Anderson acceleration is rather trivial. Each rank simply computes its n/p entries of $W\alpha$. No communication is necessary. The cost is $O(nm/p)$.

Dense Multiplication $J^{-1,(N)} \cdot V$ on Distributed Data (JV) For generalized Broyden, the numerical kernels are more involved than for Anderson acceleration, since the Jacobian is computed explicitly.

The update formula is based on two dense matrix multiplications, the first being $J^{-1,(N)} \cdot V$. I do not consider the trivial subtraction in $\widetilde{W} = W - J^{-1,(N)} V$. Both, the current and the previous Jacobian are distributed column-wise. V features a row-wise decomposition, as stated above. For this first dense matrix multiplication, each rank computes a local contribution of the full size of $\widetilde{W}$. All contributions are then summed up in an allreduce step. Afterwards, the result needs to be scattered again such that each rank holds its rows of $\widetilde{W}$. Figure 72 visualizes this operation. The realization in preCICE applies this scheme per entry of $\widetilde{W}$: instead of computing full size local matrices $\widetilde{W}$, preCICE computes a distributed dot product for every entry of $\widetilde{W}$. This alternative has a higher communication overhead since smaller messages are communicated, but needs far less memory. Only the local part of $\widetilde{W}$ needs to be stored at every rank. The computational cost is $O(n^2 m/p)$ for the local matrix multiplication and $O(nm \log p)$ for the reductions.

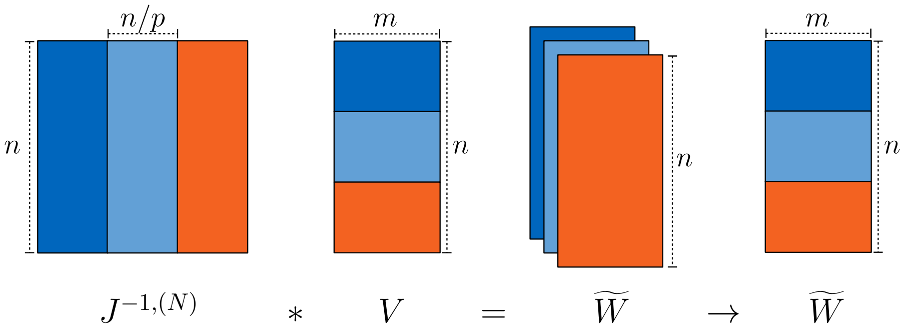


Figure 72: Schematic view on the first matrix multiplication $\widetilde{W} = J^{-1,(N)} \cdot V$ **(JV)** of the inverse Jacobian update on distributed data. A similar figure is also used in [43].

Dense Multiplication $\widetilde{W} \cdot Z$ on Distributed Data (WZ) The second dense matrix multiplication of the generalized Broyden update is more complex than the first one as it results in a matrix of Jacobian-type, more concretely the Jacobian update $\Delta J^{-1} \in \mathbb{R}^{n \times n}$, which is distributed column-wise. To compute this Jacobian update matrix, every rank needs every block from every rank. Figure 73 makes this last sentence easy to understand. To realize this, preCICE uses a memory efficient scheme, which computes the single contributions step-by-step in p so-called cycles. Therefore, additional communication channels let every rank communicate with its left and right neighbor, so to speak with $\tilde{p} - 1$ and $\tilde{p} + 1$. The basic master-slave communication does not allow for such communication, two further communications are established if generalized Broyden is used as acceleration method. Please recall Figure 62 in Section 4.2.2 for an overview on the communication architecture in preCICE.

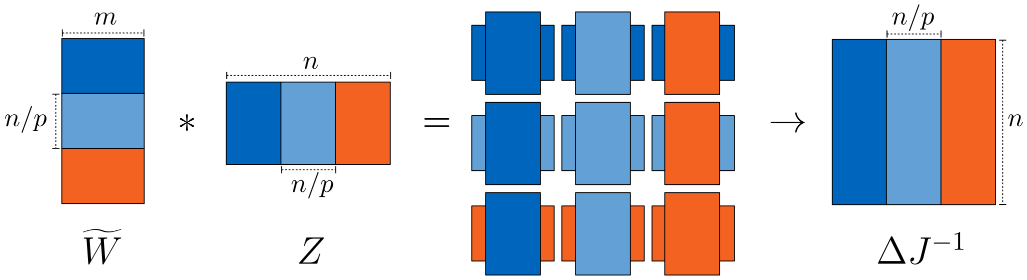


Figure 73: Schematic view on the second matrix multiplication $\Delta J^{-1} = \widetilde{W} \cdot Z$ **(WZ)** of the inverse Jacobian update on distributed data. A similar figure is already used in [43].

The cyclic procedure now works as follows: each rank starts by multiplying its local $\widetilde{W}$ with its local Z. Afterwards, it sends $\widetilde{W}$ to its right neighbor and receives accordingly a new $\widetilde{W}$ from its left neighbor. Multiplying this new $\widetilde{W}$ with the local Z gives the next contribution. After p cycles, every rank computed all contributions. In particular, all necessary contributions are already available locally. No additional re-distribution is necessary. Visualizing this procedure simplifies the understanding drastically: Figure

74 gives an example with $p = 3$, where each cycle is marked in a different color. This procedure allows for an elegant overlay of computation and communication: in each cycle, each rank first opens a receive buffer for the new $\widetilde{W}$, then sends asynchronously the old $\widetilde{W}$ and computes the multiplication with the old $\widetilde{W}$ just afterwards. Each local matrix multiplication has a complexity of $O(m \cdot n/p \cdot n/p)$, which results in total computation effort of $O(n^2 m/p)$. In theory, neglecting latencies, the communication needs $O(n/p \cdot m \cdot p) = O(nm)$, hence independent of p. For small messages, however, a linear dependence on p is more realistic. Due to the overlay of computation and communication, the total cost reads $\max(O(n^2 m/p), O(nm))$.

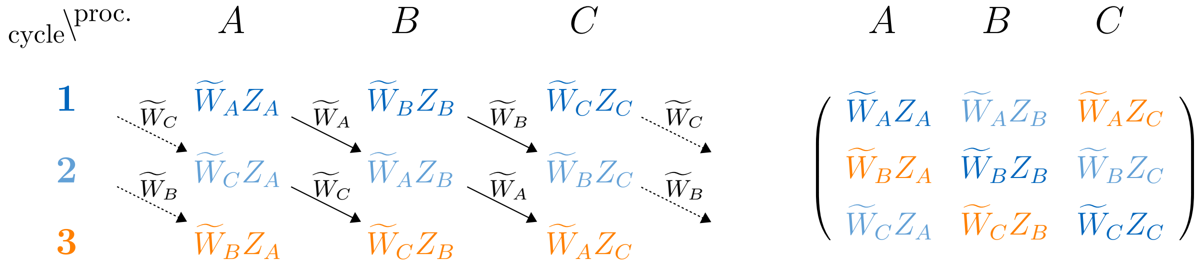


Figure 74: Cyclic scheme for the dense matrix multiplication $\Delta J^{-1} = \widetilde{W} \cdot Z$ **(WZ)**. The figure shows an example with three processors A, B, C. The sub-matrices such as $\widetilde{W}_A Z_B$ are colored by their respective cycle, not by their processor. Arrows indicate communication. The left sketch shows the computation and communication order whereas the right sketch shows the storage location. A similar figure is used in [43].

Compute Update $\Delta\tilde{x} = -J^{-1}R^k$ on Distributed Data (JR) The last kernel for the generalized Broyden method is the computation of the update $\Delta\tilde{x} = -J^{-1}R^k$. J^{-1} is decomposed column-wise whereas the residual R^k is decomposed row-wise. Hence, each rank computes a local contribution, followed by an all-reduce step and a final scattering of the resulting update vector. This operation simply corresponds to step **(JV)** if V has only a width of one column, compare Figure 72. The local block has a cost of $O(n^2/p)$, resulting in a total cost of $O(n^2/p) + O(n \log p)$

Kernel	Operation	Anderson Acceleration	Generalized Broyden
QR	update $V = QU$	$O(\frac{nm^2}{p}) + O(m^3) + O(m \log p)$	$O(\frac{nm^2}{p}) + O(m^3) + O(m \log p)$
BS	solve $U\alpha = -Q^T R^k$	$O(\frac{nm}{p}) \ + O(m^2) + O(m \log p)$	—
MBS	solve $UZ = Q^T$	—	$O(\frac{nm^2}{p})$
WA	$W\alpha$	$O(\frac{nm}{p})$	—
JV	$\widetilde{W} = W - J^{-1,(N)}V$	—	$O(\frac{n^2 m}{p}) + O(nm \log p)$
WZ	$\Delta J^{-1} = \widetilde{W}Z$	—	$\max(O(\frac{n^2 m}{p}), O(nm))$
JR	$J^{-1}R^k$	—	$O(\frac{n^2}{p}) + O(n \log p)$
Total	—	$O(\frac{nm^2}{p}) + O(m^3) + O(m \log p)$	$O(\frac{n^2 m}{p}) + O(m^3) + O(nm \log p)$

Table 24: Parallel runtime complexities of the numerical kernels for both multi-secant methods, Anderson acceleration and generalized Broyden. A similar table is used in [43].

Conclusions Table 24 gives an overview of the complexity of all kernel operations. Summing up all costs gives the immediate conclusion that Anderson acceleration is significantly cheaper than generalized Broyden. The overall cost of Anderson acceleration only grows linearly with n. Furthermore, this factor always scales down with p. The most expensive part of Anderson acceleration is the QR-decomposition.

On the other hand, the generalized Broyden method scales quadratically with n and features a further linear term in n that does not scale down, but even grows logarithmically with p. Normally, both dense matrix multiplication with $O(n^2m/p)$ should dominate the total cost. To conclude, Anderson acceleration appears well-suited for massively parallel simulations as all parts that scale with mesh size n show an ideal speed-up with p. Generalized Broyden is problematic for huge meshes, but might be a valid alternative for smaller meshes and cases where Anderson needs far more columns m than generalized Broyden. Compare such comparisons in Section 3.7.4, for example. Still, generalized Broyden needs to be applied carefully. The next section gives concrete performance results. Concerning the memory requirements, generalized Broyden currently needs to store the complete Jacobian, $O(n^2/p)$ entries per rank, whereas Anderson acceleration needs no more than $O(nm/p)$ entries. Also, several parts of generalized Broyden leave room for improvement. For example, in each iteration JV and WZ would only need to compute a single column of the update. Only at the end of each timestep the full update needs be computed. As an outlook, I want to mention the possibility to exploit the low rank nature of the Jacobian to drastically reduce the memory requirement of generalized Broyden, to nearly similar level as Anderson acceleration. This then also reduces the compute complexity significantly and would make the method much better suited for massively parallel applications. Current work about this topic is documented in [177].

4.4.3 Scaling Results

For the performance measurements, I study AA(10) – Anderson acceleration with reused columns from 10 previous timesteps – and GB(0) – generalized Broyden with only the iterations from the current timesteps. These two configurations are consistent with the recommendations from Chapter 3. Both multi-secant methods are applied on the sequential (GS) system, compare Section 3.1, for a scalar data field. In this case, the size of the fixed-point equation n equals the number of vertices n, hence no inconsistency in the notation.

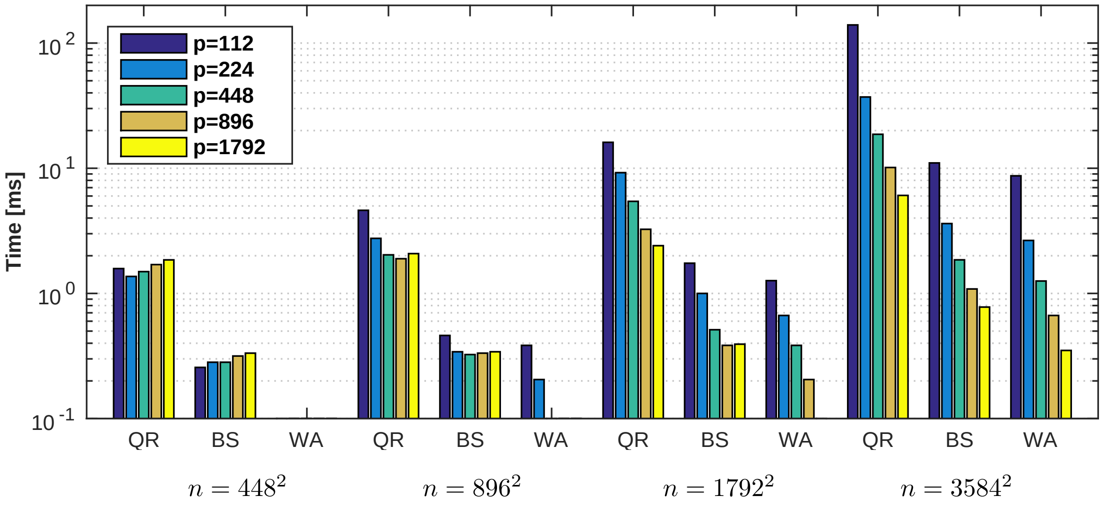


Figure 75: Scalability study for the numerical kernel of Anderson acceleration and four different meshes. For the abbreviations, compare Section 4.4.1. Suppressed bars indicate runtimes below the measurement error.

Anderson Acceleration The last section already indicates that Anderson acceleration features rather cheap operations. I, therefore, perform four strong scaling series for relatively large meshes $n = 448^2$, $n = 896^2$, $n = 1792^2$, and $n = 3584^2$. Figure 75 lists the runtimes, split up into the three numerical kernels **(QR)**,**(BS)**, and **(WA)**, as discussed in the last two sections. As expected, **(QR)** dominates for all setups. For smaller n, the measured runtimes are close to the measurement error, which leads to visible fluctuations. Here, no speed-up with p is visible as the communication part $O(m \log p)$ dominates. Where Figure 75 shows no bars at all, the runtime is below the measurement resolution of 1ms. For larger n, on the other hand, the compute part $O(nm^2/p)$ dominates, which leads to a clean speed-up with p. All numerical kernels grow linearly with n. For $p = 112$, a memory hierarchy effect is visible when increasing the mesh size from $n = 1792^2$ to $n = 3584^2$. For $p = 224$, this effect already vanishes. In total, all operations are cheap, roughly at the same size as the communication, compare Section 4.2.3, Figure 65. For $n = 3584^2$ and $p = 1792$, for example, the sum of all Anderson acceleration kernels is still below 10ms.

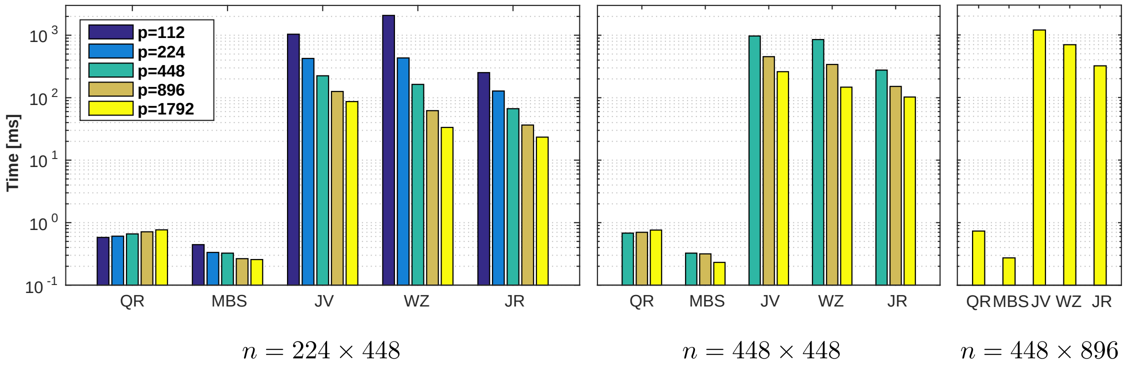


Figure 76: Scalability study for the numerical kernel of generalized Broyden and three different meshes. For the abbreviations, compare Section 4.4.1. The missing bars indicate too high memory requirements. The cyclic communication uses the `MPIPortsCommunication`.

Generalized Broyden The theoretical considerations for the generalized Broyden method, as discussed in the last section, hold in practice. Due to the higher compute and memory requirements, I use smaller meshes than for Anderson acceleration: three strong series for $n = 224 \times 448$, $n = 448^2$, and $n = 448 \times 896$. Still, the memory is not sufficient for $n = 448^2$ and $p = 112$ or $p = 224$. For $n = 448 \times 896$, only $p = 1792$ works. Figure 76 lists the numerical kernels of the remaining setups. **(QR)** and **(MBS)** appear negligible compared to the three matrix multiplications, **(JV)**, **(JR)**, and **(WZ)**. All three increase in cost with n^2 and decrease linearly with p as expected. In general, **(JV)** marks the most expensive operation. For theses results, I use `MPIPortsCommunication` as implementation of the cyclic communication (please compare Section 4.2.2). The initialization of the communication, meaning the creation of directories, publishing the connection information and establishing the connections, shows very similar results to the ones of Section 4.2.3, including similar fluctuations. If `SocketCommunication` is used as cyclic communication, however, the results differ significantly. Then, **(WZ)** cannot overlay computation and communication and the cost does, thus, increase linearly with p. For $n = 448 \times 896$ and $p = 1792$, for example, **(WZ)** needs 5.3s, which is 7.5 times more than with the `MPIPortsCommunication`.

Conclusion Anderson acceleration appears to be very well suited for massively parallel simulations. For large meshes the compute effort shows an ideal speed-up with p. This does not hold for the current version of generalized Broyden, which suffers from the dense consideration of the Jacobian. Generalized Broyden tends to be 100 to 1000 times more expensive than Anderson acceleration, despite the higher number of columns of the latter. A low rank approximation of the Jacobian for generalized Broyden seems to be a promising remedy [177]. Then, all issues, such as the cyclic communication, the high memory requirements and the $O(n^2/p)$ compute effort should disappear.

4.5 Overall Scaling Experiments

In this section, I recapitulate the two basic applications that I mention in the introduction to this thesis, Chapter 1: the Ateles Cube testcase and the turbulent FSI benchmark PfS-1a. In the introduction, I argue that classical server-based coupling concepts cannot handle either of the two applications due to the vast amount of small messages (Ateles Cube) or the tremendous asymmetry (Pfs-1a). With the parallelization concepts of this chapter, however, preCICE can successfully run both coupled scenarios on massively parallel systems. This means that preCICE does not degenerate the single physics solver's scalability.

The performance measurements of this section complement the ASTE tests described throughout the last sections. With ASTE, I test preCICE on a unit test level, meaning each feature group individually. With the two scenarios of this section, I discuss how the overall application of preCICE influences the solver's performance. This means, I compare the time spent in preCICE to the time spent in the single physics solvers in a realistic setting. Furthermore, I compare the interplay of all feature groups of preCICE with each other, to detect bottlenecks. This section shows, however, no physical results. The focus is still only on the performance. Therefore, only the first couple of timesteps of each scenario are analyzed. Chapter 5 shows physical results for various coupled applications. As the last sections already carefully compare various options for every feature group, I restrict the analysis here to the relevant options for every application. For example, the mesh repartitioning always uses the *broadcast/filter* approach, since Section 4.1.3 concludes on the superiority of this approach.

Section 4.5.1 shows performance results for the Ateles Cube. In the introduction to this thesis, Section 1.3.2 briefly introduces this testcase, showing that the old server-based parallelization concept of preCICE results in a huge overhead and in scalability limitations. For this scenario, a single Euler domain is cut into two halves at an artificial coupling interface. The coupled setup can, thus, be easily compared to the monolithic simulation. The coupling interface features matching meshes and even a matching decompositions. I, therefore, apply a simple nearest-neighbor mapping. As Ateles itself uses an explicit time integration, a parallel explicit coupling scheme is further applied. This means, in particular, the new communication concept is tested whereas the mapping and the coupling scheme are rather simple. All in all, the scenario is a perfect testcase for preCICE as a clean and easy interpretation of the results is possible.

Section 4.5.2 discusses the second testcase, the turbulent FSI benchmark PfS-1a [58]. The most important characteristic of this case is the immense asymmetry: while the fluid field has to use a very fine mesh at the coupling interface to carefully resolve the turbulent boundary layer, the structure can use a rather coarse mesh. In the introduction to this thesis, Section 1.1.2 uses this testcase exemplarily to show that coupling software that is based on a central server-like instance cannot handle such setups already on only moderately parallel systems. In contrast to the Ateles Cube, PfS-1a uses more sophisticated features of preCICE: interpolation between the non-matching grids is done with radial basis functions and an Anderson acceleration is used for the implicit coupling. This brings the case close to real applications, but makes it also harder to interpret the results. I, therefore, mainly focus on comparing the overall preCICE performance with the runtime spent in the fluid solver.

4.5.1 Ateles Cube

Scenario Description A cubical domain is cut into two equal-sized halves, orthogonal to the x-axis. preCICE is used at the artificial coupling interface, compare Figure 77. In both domains, compressible flow is simulated, governed by the Euler equations. A Gaussian density pulse is initialized in the left half and travels with constant speed towards the right half. Figure 78 shows the smooth transition over the interface. I use a high number of unknowns for Ateles, for which we know that Ateles shows a perfect strong scaling [239]. Hence, we can clearly study whether preCICE degenerates this scaling or not. Ateles uses a Cartesian tree-based mesh. For mesh level l, the complete cube holds $2^{3(l-1)}$ elements. This results in $2^{2(l-1)}$ elements at the coupling interface per side. In each element, a 12th order discontinuous Galerkin schemes is used. Thus, each element holds 12^3 Gauss points, while each Gauss point has five degrees of freedom – three for the velocity and one for the pressure and the density. At the coupling interface, this gives 12^2 vertices per element – the closest Gauss points are simply projected on the interface. The results presented in the introduction use a mesh level $l = 5$. I revisit those results just below. The further tests of this section use a mesh level $l = 6$ to allow scaling to an even higher number of cores. In principle, Ateles scales down to a single element per core if the discontinuous Galerkin order is sufficiently high [239]. Ateles uses a domain decomposition, which divides the cube, one by one, in either x, y or z direction when doubling the number of cores. This means every third time, when the domain is decomposed in x-direction, the number of cores at the interface does not change. Table 25 lists concrete numbers. Furthermore, Table 25 clearly illustrates the ratio between the number of cores at the interface to the total cores as well as the number of vertices at the interface to the total vertices. Please recall that all ASTE tests only considered vertices and cores at the coupling interface, whereas the Ateles Cube is an actual 3D scenario with a 2D interface.

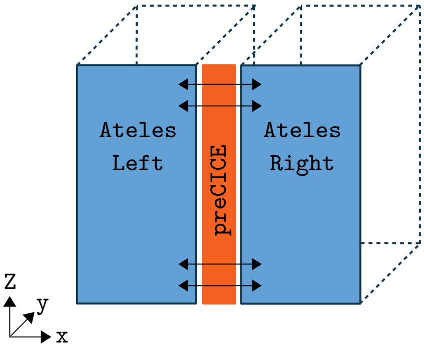


Figure 77: Ateles Cube: a cubic domain is cut into two halfes and coupled via preCICE at the artificial interface. This figure is already used in [41].

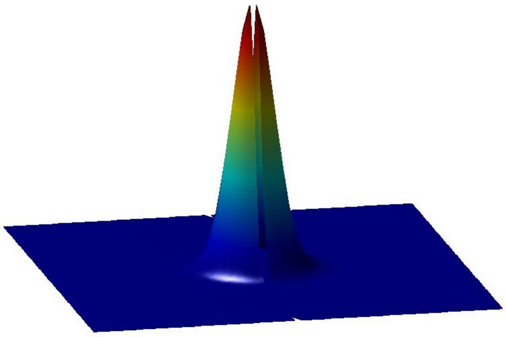


Figure 78: Ateles Cube: smooth transition of the density pulse through the coupling interface from left to right. This figure is already used in [41].

Cores	16	32	64	128	256	512	1024	2048	4096	8192
Cores at Int.	8	8	16	32	32	64	128	128	256	512
GP. p. Core	3.54e6	1.77e6	8.84e5	4.42e5	2.21e5	1.11e5	5.52e4	2.76e4	1.38e4	6.91e3
IV. p. Core	9216	9216	4608	2304	2304	1152	576	576	288	144

Table 25: Ateles Cube: number of cores at the coupling interface, total Gauss points (GP) per core and interface vertices (IV) per core for an increasing number of total cores and mesh level $l = 6$. All listed numbers are per participant.

Experiment Settings For all performance results, I compute 10 timesteps, but, contrary to ASTE, no coupling iterations. I use MPI as underlying implementation for the master-slave communication as well as for the M:N communication, compare section 4.2. A nearest-neighbor mapping is used as interpolation method. All Ateles Cube tests were run on the thin nodes partition of SuperMUC phase 1, since the 16 cores per node fit very well to the octree data structure of Ateles. Next, I first recapitulate the results from the introduction, on mesh level $l = 5$, and afterwards I discuss the initialization and work per timestep for mesh level $l = 6$ in depth.

Recapitulation of the Introductory Experiment In the introduction, Section 1.3.2 compares the runtime of the monolithic Ateles Cube with the coupled variant that uses the old server-based parallelization concept of preCICE, compare Figure 7. This server-based concept has several deficiencies, which the introduction discusses in detail. I reuse this figure now, complemented by the runtimes for the new fully parallel concept of this chapter, see Figure 79. Almost no overhead is visible compared to the monolithic run. In particular, preCICE shows no influence on the scalability of Ateles. Furthermore, also the initialization time improves significantly. In the old server-based concept, the initialization is dominated by the compute effort of the nearest-neighbor mapping. This step is now computed on distributed data and hence negligible. The initialization time improves roughly by a factor of 10 resulting in approximately 5s for the new concept, depending on the number of cores. The problem, which I pose in the introduction, is solved. Even the needed resources are smaller, since no separate nodes for the servers need to be reserved. The next paragraphs analyze the performance in detail.

Initialization for Mesh Level $l = 6$ Figure 80 lists the initialization timings for a strong scaling study. The mesh level l is 6 and the number of cores per participant p is doubled nine times from 16 to 8192. $p = 16384$ showed problems when establishing the master-slave connection via `MPI_COM_SPLIT`.

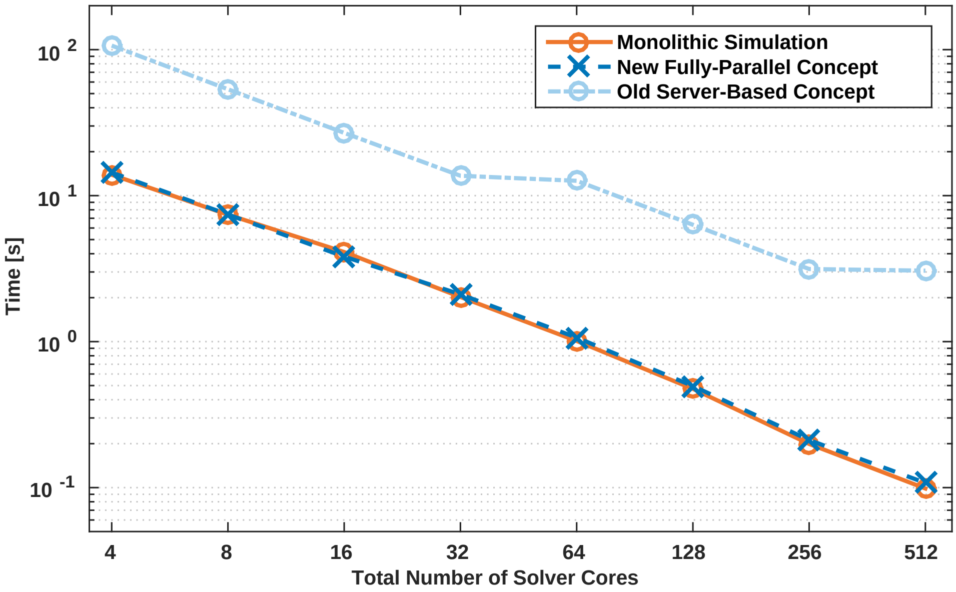


Figure 79: Work per timestep for the Ateles Cube, strong scaling for mesh level $l = 5$. A monolithic simulation is compared to the old server-based parallelization concept of preCICE and to the new fully-parallel concept. The same figure, but without the new concept is already used in the introduction to this thesis, Figure 7. For the old server-based concept, the resources for both server processes are neglected.

This is no consistent problem, compare the successful runs in [181]. All timings are measured at the participant who re-partitions the communicated mesh. Without loss of generality, this is `Ateles Right`. In the following, I analyze each building block of the initialization step by step. First, Figure 80 lists the initialization time of Ateles itself. These timings include the time spent in the Ateles-preCICE adapter as well. For a lower p, a logarithmic increase with p is visible, overlaid by several outliers. For a higher p, the increase with p becomes significant. The main purpose of showing theses timings here is to compare them to the overall preCICE initialization, which is just the next block in Figure 80. For the preCICE initialization, I sum up the time spent in `initialize` and `initializeData`. The initialization time decreases till $p = 128$, and increases afterwards. Starting with $p = 2048$ and a timing of roughly one minute, the initialization becomes significant. Therefore, I have a careful look which building block causes this increase. Establishing the master-slave connection is insignificant for a small amount of MPI ranks. For a higher amount, the runtime appears, however, to grow exponentially with p and quickly becomes a severe bottleneck. Creating the geometries, including the re-partitioning of the communicated mesh, is discussed in detail in Section 4.1.3. For the Ateles Cube, it scales very well for small p, where the compute effort dominates. For a higher p, the feedback step dominates, which lets the runtime increase with p. For $p = 8192$, the runtime is still below 10s, which makes it tolerable. Initializing the mapping is a pure compute effort for the nearest-neighbor mapping only using local mesh decompositions. This step scales, thus, quadratically with p, while the earlier discussed domain decomposition steps, compare Table 25, are clearly visible. Thus, for a small p the effort for initializing the mapping can be significant, while it is negligible for larger p. *Build VD/CM* summarizes the three steps that prepare the connection data for the M:N communication: communicate the vertex distributions, broadcast the vertex distributions, and compute the communication maps from the vertex distribution, compare Section 4.2.1. For smaller p, again the compute effort dominates leading to good scalability, while for larger p, the communication effort lets the runtime increase. However, this step is always below 1s. Before the actual M:N connections are established, sub-directories are created to speed up the I/O access to the file system. As already discussed in Section 4.2.3, this step grows super-linearly with p. Establishing the M:N communication itself grows linearly with p. Section 4.2.3 analyzes that the effort is mainly due to the internal MPI effort, not the file system access. An M:N communication built on the TCP/IP sockets implementation would significantly speed up this step.

I conclude that the MPI operations for the master-slave communicator as well as for the M:N communicators show a weak performance. The creation of the directories and the file system access show the expected behavior. All these steps are fully tolerable for medium sized parallel setups, but get significantly expensive for a really high number of cores. Alternatives for this range are necessary. All other building blocks, such as the mesh-repartitioning, the initialization of the mapping, or the building of the communication maps are all fully tolerable, especially because they are faster than the Ateles initialization.

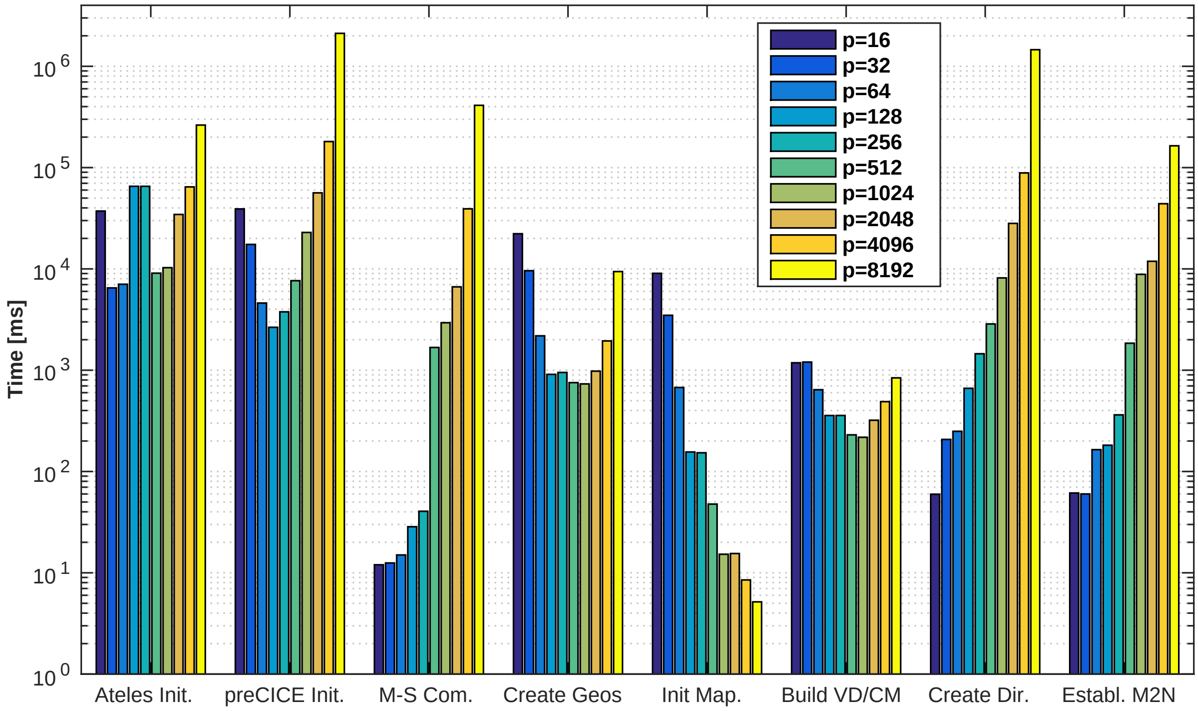


Figure 80: Ateles Cube: strong scaling for the initialization timings, mesh level $l = 6$.

Work per Timestep for Mesh Level $l = 6$ After the preliminary results for mesh level $l = 5$ further above, I now have a look at the work per timestep for mesh level $l = 6$. Typically, the work per timestep is of higher importance than the initialization as a huge amount of timesteps is needed compared to the single effort in the initialization. This is particularly true for Ateles, as the explicit timestepping implies a small timestep size. Figure 81 lists the strong scaling results for mesh level $l = 6$. I compare the overall simulation time per timestep to the time spent in every advance call. It is clearly visible that the preCICE effort is negligible. The Ateles scaling shows the expected perfect linear behavior. Only from $p = 4096$ to $p = 8196$ a small drop is visible. The time spent in advance is mainly used to synchronize the solver ranks to make up for a non-ideal load-balancing, possibly due to the time spent in the Ateles-preCICE adapter. This is obvious, when looking at the actual communication and mapping effort that preCICE needs. The main goal of the new parallelization concept of preCICE is achieved: the scalability of Ateles is not degenerated by preCICE.

4.5.2 PfS-1a Benchmark

After the Ateles Cube scenario in the last section, which still marks a rather artificial testcase, this section discusses a more complex application. De Nayer et al. propose a turbulent FSI benchmark in [58], named PfS-1a[46]. For this testcase, I apply sophisticated coupling methods such RBF mappings for the non-matching meshes and quasi-Newton post-processing for the strongly-coupled FSI problem. This brings the testcase closer to real applications, but makes the performance results also harder to analyze. All in all, this section perfectly complements the results from the last section.

A flexible rubber plate is attached to a rigid cylinder and excited under turbulent flow at a Reynolds number of 30,400. The geometry is very similar to the classical FSI benchmarks from Turek et al.

[46] http://qnet-ercoftac.cfms.org.uk/w/index.php/UFR_2-13

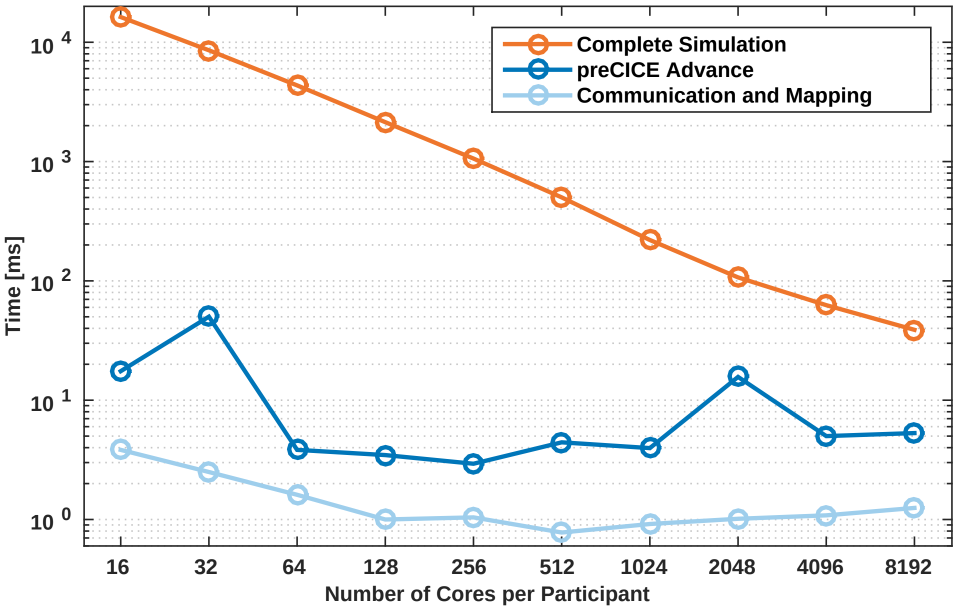


Figure 81: Ateles Cube: strong scaling for the work per timestep, mesh level $l = 6$.

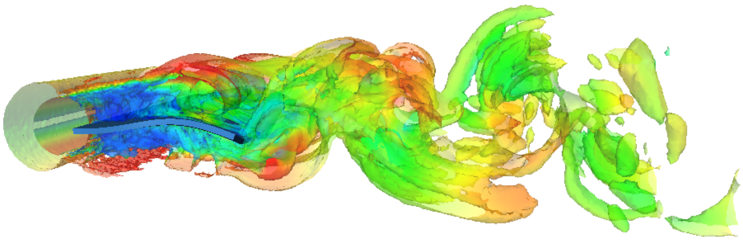


Figure 82: Screenshot of the Pfs-1a benchmark: structural deformation and iso-surfaces of the vorticity. Please note, that the structure is not fixed in the span-wise direction such that the deformation is over-predicted compared to [58].

[199] (see also Section 3.7.1), but the turbulence changes the picture quite significantly: the simulation is inherently 3D, the fluid mesh needs a much finer resolution, and also the FSI coupling behaviour changes. The scenario further increases in cost as statistical data needs to be collected over a wide range of timesteps in order to compare to experimental data. To get a physical impression, Figure 82 shows a screenshot of the flow field around the excited structure. The physical results are, however, only preliminary. The variance in the deformation in span-wise direction is over-estimated, probably due to different boundary conditions compared to [58]. Just as for the Ateles Cube, I do not consider physical results here. The purpose of this section is solely to evaluate the performance of preCICE. Section 5.4 gives physical results for a turbulent FSI case. To test the performance, I re-use exactly the same fluid mesh as in the original work [58] – therein refered to as subsetcase mesh. The fluid mesh consists of roughly 13 million hexahedral elements. As physical solvers, I use the Alya modules Nastin and Solidz, compare Section 2.4. As the normalized fluid mesh width in wall direction y^+ is always below 0.8, I resign to use any additional LES model besides the subgrid scale finite element method in Nastin [114]. The structural mesh only counts 18,792 hexahedral elements. This asymmetry influences the computational resource for every domain drastically. Whereas many resources can be applied in the fluid domain, the structure domain only needs a few cores. Figure 83 shows the domain decomposition at the coupling interface for 6,267 fluid partitions and 3 structure partitions. Next, I recapitulate the basic reasoning about the PfS-1a benchmark from the introduction. Afterwards, I analyze the performance for the initialization as well as per timestep.

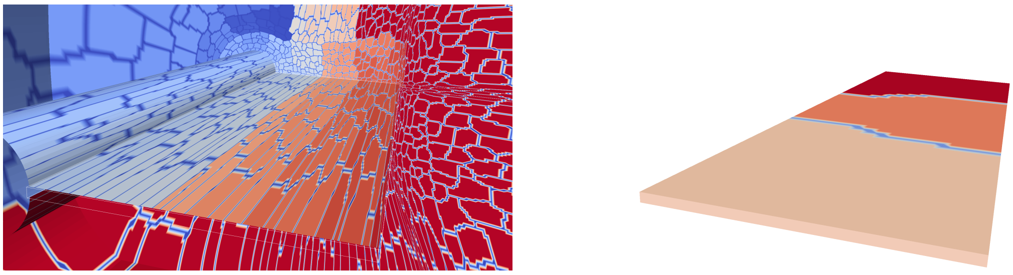


Figure 83: PfS-1a benchmark: decompositions of the coupling surface for both single physics solvers, Nastin(left) and Solidz(right) for 6,268 and 4 cores, respectively. 321 decompositions of Nastin and all three decompositions of Solidz, besides the empty master rank, touch the coupling surface.

Recapitulation of the Introduction In the introduction to this thesis, in Section 1.1.2, I discuss why cases with a very high cost asymmetry suffer in particular from coupling concepts that use a central server-like instance. In [57], the scalability for the PfS-2a benchmark, with the same mesh that I use in this section, was restricted to less then 100 fluid cores. This is unsatisfactory as the fluid solver should surely scale beyond this threshold. Figure 84 shows scaling results for a pure fluid simulation of the PfS-1a benchmark with Alya. The work per timestep shows still a speed-up at 6,272 cores and gets close to the costs of the structure solver at 4 cores. The goal of the parallelization of preCICE in this chapter is to not destroy this fluid scalability when coupled to a structure solver.

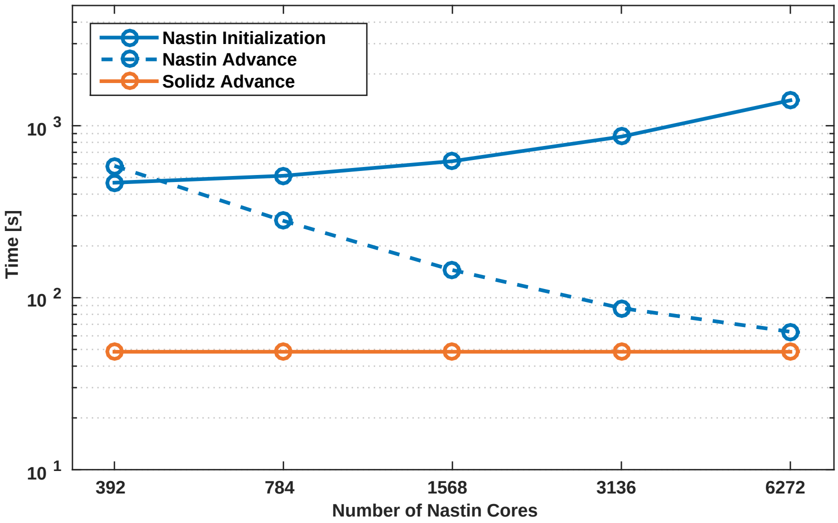


Figure 84: PfS-1a benchmark: strong scaling results for a single-physics fluid simulation, split into the initialization and the work per timestep (advance). Furthermore, for comparison, the work per timestep for the pure structural problem on 4 cores is plotted.

The overall mesh asymmetry also induces an asymmetry at the coupling interface. The coupling mesh of the fluid solver counts 63,321 vertices while the structure solver only counts 4,960 vertices. This is still a moderate discrepancy. A membrane structure solver with more sophisticated structural elements could use even much less interface vertices. Coupling approaches with a central instance typically have to communicate both meshes to the central instance to compute the mapping there. This communication overhead limits the fluid scalability. With the fully parallel concept of this chapter, we can freely choose at runtime which mesh to communicate. Thus, we choose to only communicate the small structure mesh from the structure solver to the fluid solver and compute the interpolation at the fluid solver, distributed over all interface ranks. Furthermore, we can choose the structure solver as the acceptor of the M:N communication, compare Section 4.2.1. Thus, preCICE only needs to publish connection information for every structure rank.

Table 26 collects statistics on the asymmetry. It becomes obvious that the decomposition of the fluid solver is not optimized towards the coupling interfaces, but towards its domain. At the interface, the number of vertices per rank varies between 9 and 402. Furthermore, Table 26 also shows statistics on the communication layout, where it compares the applied radial basis function (RBF) mapping to a nearest neighbor mapping. The overlap regions for the RBF mapping are significantly higher than for the nearest neighbor mappings. The communicated data increases by a factor of 10.

	Nastin			Solidz		
	min	max	avg	min	max	avg
Interface Vertices	9	402	197.26	1652	1654	1653.3
Com. Partner (NN)	1	2	1.396	135	172	149.3
LCM $\sum_q \lvert\Psi^{\tilde{p}}(q)\rvert$ (NN)	3	111	31.016	3252	3406	3318.7
Com. Partner (RBF)	1	3	1.508	136	188	161.3
LCM $\sum_q \lvert\Psi^{\tilde{p}}(q)\rvert$ (RBF)	112	728	340.38	38061	35236	36420.7

Table 26: Pfs-1a benchmark: coupling surface statistics for 6,268 fluid cores and 4 structure cores (leading to 321 and 3 cores at the interfaces). The coupling surfaces count 63,321 and 4,960 for the fluid and structure solver, respectively. This table shows the structure mesh decomposition for two different mappings: nearest neighbor (NN) and radial basis function (RBF). The local communication map (LCM) $\Psi^{\tilde{p}}$ list the vertices of rank $\tilde{p}$ that it has to communicate to every rank of the other participant, compare also Section 4.2.1.

Experiment Settings For the performance measurements, the number of fluid cores is increased step-wise p_F = 780, 1564, 3132, 6268, while the number of structure cores is fixed at $p_S = 4$. All measurements are done on the SuperMUC Haswell partition, compare the introduction to this chapter. The fluid solver starts from a pre-computed fluid simulation of 10 timesteps. Afterwards, I only measure the first coupled timestep while allowing 8 FSI coupling iterations. As the fluid solver is not yet in a stable convergent state, neither is the FSI coupling. This is unfortunate, but does, however, not significantly influence the performance measurements. Furthermore, as mentioned already above, Section 5.4 discusses physical results of another, yet very similar turbulent FSI application. Contrary to all other results of this chapter, I do not perform five, but only one single run per setting as fluctuations only play a minor role here. The RBF mapping uses compact thin plate splines as basis function with a support radius of $r = 0.005$, covering two elements in both orientations for the flow and span-wise direction and 20 elements in both orientations for the wall direction. The RBF system is solved up to relative tolerance of 10^{-5}. I apply the parallel Anderson acceleration J-AA as quasi Newton method using columns of all 8 iterations, a QR1 filter with $\epsilon = 10^{-6}$ and the residual-sum preconditioner. Since I only measure the first timestep, the amount of columns of the Anderson acceleration underestimates a realistic setting to some extent. As the cost of Anderson acceleration grows linearly with the number of columns, a more realistic cost can, however, be easily extrapolated. Furthermore, the quasi Newton costs are rather insignificant for this application as I discuss further down. Both, the fluid and structure solver, use a non-linear convergence criterion of 10^{-5} with 10 and 5 maximum iteration, respectively, accordingly tuned linear solver settings, and a timestep size of $\Delta t = 5 \times 10^{-4}$ s. This leads to a computational time of slightly over a minute per fluid timestep for 6,268 cores, compare Figure 84.

Computational Costs of the Initialization Figure 85, left, shows the computational cost of the initialization. The overall preCICE initialization time increases with the number of fluid cores, but is always below 10s and therefore insignificant compared to the fluid solver's initialization of over 20 minutes for 6,268 cores, compare Figure 84. Establishing the master-slave communication increases, as expected, with the number of cores, similar to the mesh creation on the fluid side. The latter is dominated by the global feedback step. To setup the M:N communication is a very cheap operation for this scenario as only three communications, one per structure rank, need to be set up. Again, the fully parallel layout of preCICE can take advantage of the scenario's asymmetry. Initializing the coupling scheme has insignificant costs, especially for $p_F = 3132$ and $p = 6268$, where a memory hierarchy effect lets the runtime drop below the measurement error. The cost to setup the RBF mapping remains below 1s for all four fluid core counts.

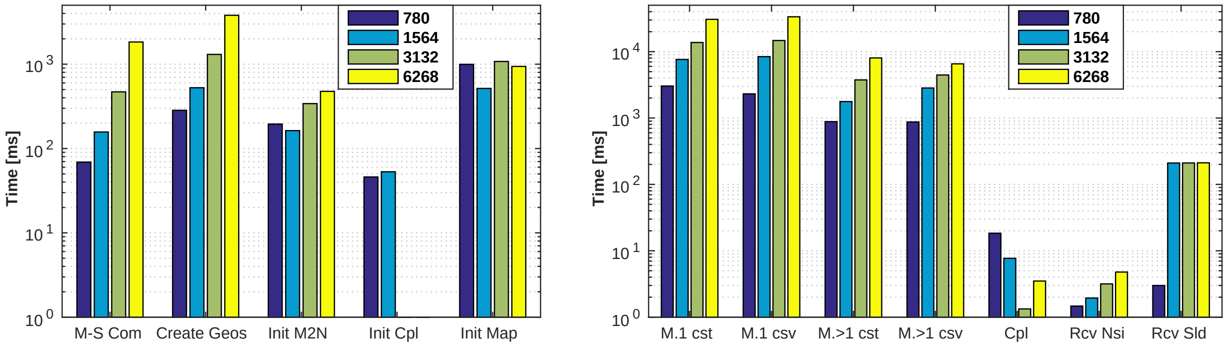


Figure 85: Pfs-1a benchmark: initialization (left) and work-per-timestep (right). The setup of the master-slave communication (M-S com) and the geometry creation are measured for Nastin. The setup of the M2N communication for Solidz. M.1 cst: first consistent mapping, M.>1 csv: average of all conservative mappings after the first one. Rcv Nsi: receive at Nastin, Rcv Sld: receive at Solidz.

Computational Costs per Timestep Figure 85, right, shows the computational cost per timestep. The overall preCICE cost is fully dominated by the RBF mapping, but still below the cost for the fluid timestep. Please note that per timestep, six RBF mappings need be applied, one for each dimension for the forces and the displacements. Similar to RBF measurements of ASTE, Section 4.3.3, I distinguish between the first mapping computation and all others, since only the first one includes the computation of the preconditioner. Both differ roughly by a factor of two to five. I further distinguish between the consistent mapping and the conservative mapping. The latter shows an only slightly higher cost. In general, the cost per mapping increases significantly with the number of cores. This is not due to a higher number of iterations, which, for example, is approximately 2,550 for $p_F = 780$ and 2,320 for $p_F = 6268$, averaged over the first 12 mappings. The increase is caused by the significant communication overhead of the polynomial rows. Still, the amount of iterations leaves plenty of room for improvement.

The computational cost of the coupling schemes shows an ideal speed-up. Please note that the applied residual-sum preconditioner requires a full new QR decomposition per iteration, compare Section 3.6.3. Still, even if we account for the limited amount of used columns by multiplying the runtimes by a factor of, let us say, 20, the effort still remains negligible. The reason is that the coupling scheme can take full advantage of the asymmetry as it only needs to be computed on the coarse coupling mesh of the structure solver. A brief remark: for $p_F = 3132$, the fluid solver crashes after 4 iterations, causing a lower average number of columns and thus, causing the lower runtime for the coupling scheme. To discuss the communication time, Figure 85 shows the receive time for both participants. The receive time at the fluid solver is very small as every fluid rank only needs to receive relatively small messages, compare Table 26 for average values on approximately 340 vertices for $p_S = 6269$. Furthermore, the receive time increases linearly with p_F, as the amount of asynchronous send operations per structure rank increases. The receive time for the structure solver, on the hand, is substantially higher as each structure rank needs to receive values on approximately 36,421 vertices in average. This size does not change with p_F, hence neither does the runtime. Only for $p_F = 780$, a smaller receive time is measured, probably due to the lower amount of messages.

Conclusions The fully parallel concept of preCICE allows to take advantage of the immense asymmetry of the PfS-1a benchmark. We can restrict the communication and the quasi-Newton computations to the coarser structural mesh. Therefore, both steps have almost no influence on the overall performance. Also, the initialization effort can take advantage of the asymmetry: it is sufficient to only re-partition the coarse mesh and the M:N communication only needs to establish one communication per structure rank. The RBF mapping, on the other hand, results still in significant work. The runtime increases with the number of fluid cores, as the polynomial rows lead to global communication. Even if restarting from previous iterations or more sophisticated preconditioners ameliorated the number of iterations significantly, an RBF method without a global polynomial is needed. Still, the goal of this chapter is reached: even for 6,268 fluid cores, the time spent in preCICE is smaller than the time spent in the fluid solver. Hence, a scaling to 60 times more cores than for a server-based coupling software such as in [58] is possible.

Summary of Chapter 4

- The mesh re-partitioning is based on a gather-scatter procedure through the master rank, which remains tolerable since only necessary during initialization. Two variants are supported: the *broadcast/filter* variant, which is more efficient for most setups, and the *pre-filter/post-filter* variant, which is an alternative for memory critical setups.
- The parallel communication reuses the existing 1:N communication as kernel to build up a general M:N communication, avoiding deadlocks at initialization. An asynchronous kernel communication further avoids deadlocks during the actual communication. The implementation clearly separates the kernel implementation from the logical M:N layer.
- Initialization of the communication remains tolerable up to 1,000 interface cores. Afterwards, the MPI routines as well as the publication strategy, currently based on files, become bottlenecks. In general, the MPI initialization routines (`MPI_Comm_accept`, `MPI_Comm_connect`, etc.) are not robust and do not fully support the MPI standard for the INTEL and the IBM implementation.
- MPI and TCP/IP are both very efficient for the actual M:N communication, although MPI outperforms TCP/IP by an approximate factor of seven.
- The four data mapping variants read/write - consistent/conservative are restricted to the two cases read-consistent and write-conservative to allow for an easier parallelization without much practical limitation.
- The parallelization of projected-based mappings, then, reduces to a simple sorting at initialization without any communication during each mapping step. The costs are negligible.
- The parallelization of the RBF mapping is based on PETSc and shows performance limitations due to the non-optimal polynomial rows.
- For the multi-secant methods, the QR-decomposition is changed to an update scheme, which is more efficient and allows for an easier parallelization.
- Therewith, the Anderson acceleration is well-suited for massively parallel execution, since the compute effort scales well with the interfaces cores. The generalized Broyden method, however, suffers from the dense consideration of the Jacobian, which limits the method's applicability. A low-rank approximation should solve this problem.
- All parallelization concepts are tested with two overall scaling experiments. The Ateles Cube scenario uses an explicit coupling scheme and a projection-based mapping. The initialization remains tolerable for medium-sized parallel setups, but becomes problematic for highly-parallel setups. Per timestep, the coupling hides completely behind the solver's costs, not deteriorating the overall scalability.
- The PfS-1a scenario features a very asymmetric setup and uses the Anderson acceleration besides RBF mapping schemes. Since communication and the Anderson acceleration are both performed on the coarse structure coupling mesh, they are negligible. The RBF mapping is significant in cost, but the coupling remains cheaper than the solvers' costs till 6,268 fluid cores.

5 Show Cases

In Chapter 3, I introduce an inter-solver parallel layer into FSI simulations by means of parallel coupling schemes. Tests with standard FSI benchmarks show that the convergence speed compared to sequential coupling schemes is not degenerated. In Chapter 4, I further introduce parallelism on an intra-solver level by porting preCICE to a fully parallel peer-to-peer layout. Performance tests on a package level and for simple coupled simulations show that legacy codes can now be coupled without decreasing their scalability. Thus, both parallel layers do work. But do we really need them? I answer this question by applying both parallel layers in realistic scientific applications in this fifth chapter. My focus thereby is not on the individual application, but on the performance and applicability of the parallelization concepts. Furthermore, the very heterogeneous range of applications and coupled solvers shows the inherent flexibility of the partitioned approach, but also of the software concepts in preCICE. Just next, in Section 5.1, I give a brief overview on all showcases of this chapter. Sections 5.2 to 5.6 then detail all five showcases one by one and can be read in any order.

5.1 Overview on Show Cases

Table 27 collects all showcases, the applied solvers, and the applied methods. In following, I briefly summarize the main motivation for each case.

Section	Case	Physics	Solvers	Cpl. Scheme	Mapping
5.2	Aorta	FSI – Hemodynamics	Nastin, Solidz	Various	Matching
5.3.2	Jet	Flow & Acoustics	Ateles	J-EX/J-EX	Matching
5.3.3	Bending Tower	FSI & Acoustics	OpenFOAM, Ateles	J-EX/J-AA	NN
5.4	Hemisphere	FSI – Aeroelasticity	Nastin, Carat++	J-AA	NN
5.5	Multi-Cylinder	FSI – Aeroelasticity	SU2, Solidz	M-AA	Matching
5.6	UFSI3	FSI & UQ	Nastin, Solidz	J-AA	Matching

Table 27: Overview on all showcase of Chapter 5.

Aortic Blood Flow – Section 5.2 The Aorta showcase gives an example for blood flow through a realistic aortic geometry. The use of surrogate boundary conditions does not allow to draw physical conclusions, but does not interfere with the numerical relevance. The purpose of this showcase is twofold. First, I want to show that a partitioned approach with quasi-Newton coupling can deal with such a challenging setup in an efficient way while Aitken underrelaxation fails to do so. Second, I compare the runtime of serial and parallel coupling schemes to show that the inter-parallel layer leads indeed to a lower total simulation time while also using less resources. The Aorta showcase is joint work with Juan Carlos Cajas et al. from the Barcelona Supercomputing Center. The geometric setup is provided by Jordi Martorell et al.

Fluid-Structure-Acoustics Interaction – Section 5.3 Fluid-structure-acoustics interaction (FSAI) enriches classical FSI by resolving acoustic effects, which results in a challenging three-field coupling. This section consists of two scenarios: a three-field flow coupling for a 2D subsonic jet and a full FSAI for a 3D bending tower in cross flow. I use both scenarios, in particular, to show that preCICE is able to technically handle such complex three-field couplings in an efficient way. Here, the Jet scenario uses two explicit coupling schemes whereas the Bending Tower scenario composes an implicit coupling between fluid and structure with an explicit coupling between fluid and acoustics. All combinations lead to an inter-solver parallelism, generalizing the ideas from Chapter 3. I study the load balancing between three solvers exemplarily for the Jet scenario. This showcase is joint work with David Blom, Delft University of Technology and Verena Krupp, University of Siegen, besides all other collaboration partners in the ExaFSA project. For the Bending Tower scenario, we already published results in [25].

Turbulent Flow around an Elastic Hemisphere – Section 5.4 As the name of the section says, this showcase simulates a turbulent flow around an elastic hemisphere, which is mounted on a no-slip bottom. Similar to the PfS-1a testcase, which I used in Section 4.5.2 for scalability tests, the Hemisphere showcase also features a very high cost asymmetry: the fluid solver uses a very fine mesh at the coupling interface to carefully resolve the turbulent boundary layer, while the structure solver is very cheap and can even be computed in serial. Compared to the PfS-1a testcase, the fluid domain even holds three times more degrees of freedom. Preliminary results for the Hemisphere showcase show the successful application of the intra-parallel layer developed in this thesis as well as the quasi-Newton coupling schemes – despite the turbulent flow. The showcase is joint work with Aditya Ghantasala et al. from the Chair of Structural Analysis, TUM, and Juan-Carlos Cajas, Herbert Owen, et al. from the Barcelona Supercomputing Center. I borrow the setup of the fluid solver from [227], where benchmark results for a pure fluid simulation are compared to experimental data. Guillaume de Nayer provided us with the fluid mesh from the original setup.

Simulation of a Brush Seal – Section 5.5 The Multi-Cylinder showcase studies flow around multiple thin and elastic cylinders – a simplified model of a brush seal. Each cylinder is simulated by a separate structure solver. The purpose of the simplified model is to study how the amount and the arrangement of the cylinders influence their vibrations. The main focus of the showcase is to show the applicability of the multi-coupling methodology developed in Section 3.8 for such a real application. This showcase is joint work with Alexander Rusch in collaboration with Alexander Fuchs et al. from the Chair of Turbomachinery and Flight Propulsion, TUM. First results are already reported in the bachelor thesis of Alexander Rusch [172].

Uncertainty Quantification of the FSI3 Benchmark – Section 5.6 The UFSI3 showcase takes the plain FSI3 benchmark from Turek et al. [199] and studies how uncertainty in the physical input parameters propagates through the model. A sparse grid collocation methodology leads to a set of independent deterministic simulations. These simulations can be computed in parallel resulting in an additional third parallel layer. The main purpose of this showcase is neither original research in uncertainty quantification, nor the concrete scenario, but to show how the three parallel layers interplay. I show that the flexible software environment provided by preCICE can deal with such complex setups in a rather simple way. The showcase is joint work with Ionut Farcas. Preliminary results are already reported in his master thesis [79]. The complete results of this showcase with a focus on the UQ methodology is further prepared for publication in [80].

Remark: On several occasions throughout this chapter, my style of writing switches from a single-author perspective to a group perspective to emphasize joint work. In such a case, *we* encompasses the just mentioned respective collaborators besides myself.

5.2 Aortic Blood Flow

In Chapter 3, I develop parallel FSI coupling schemes, whose performance I compare to serial coupling schemes by simply looking at the number of iteration needed until convergence. The underlying assumption is that if serial and parallel coupling schemes both need the same amount of iterations, parallel coupling schemes result in a speed-up, since both solvers can be executed simultaneously. For a perfect load balancing between the fluid and the structure solver, a theoretical speed-up of two is possible. In this section, I use a realistic application – blood flow through an aortic geometry – to compare serial and parallel coupling schemes by means of their runtime and their needed resources. Several factors come now into play that I neglect in Chapter 3:

1. Is a perfect load balancing between the fluid and the structure solver possible? If yes, does it remain constant throughout the simulation?
2. The coupling iterations within one timestep do not show a constant runtime. For an implicit solver, typically, the first few need by far longer than the last iterations. How does this interfere with the comparison of coupling schemes solely by their iteration numbers?
3. The fixed-point acceleration scheme itself also includes a compute effort that differs between serial and parallel coupling schemes. The parallel system usually features twice the number of rows and often also twice the number of columns of the serial system. Even more important: if the residual-sum weighting is applied, the parallel system needs a re-computation of the complete QR-decomposition in every iteration. Are all theses differences really negligible compared to the solvers' costs?

These open questions are not always easy to measure and compare. I, therefore, choose to directly compare the overall performance of the coupled simulations, which also is most important from the application perspective. To achieve the best overall runtime for any coupling scheme, I choose the resources for both solvers such that they run at their scalability limit.

As a nice side-effect of this showcase, I can show that real-world blood flow applications can be efficiently carried out by the (black-box) partitioned approach. There is still a misbelief in the FSI community that this is not possible. This misbelief, however, builds up on experiences that were made with simple Aitken underrelaxation, a decade ago. Quasi-Newton coupling schemes change the pictures tremendously as I also show further below in this section.

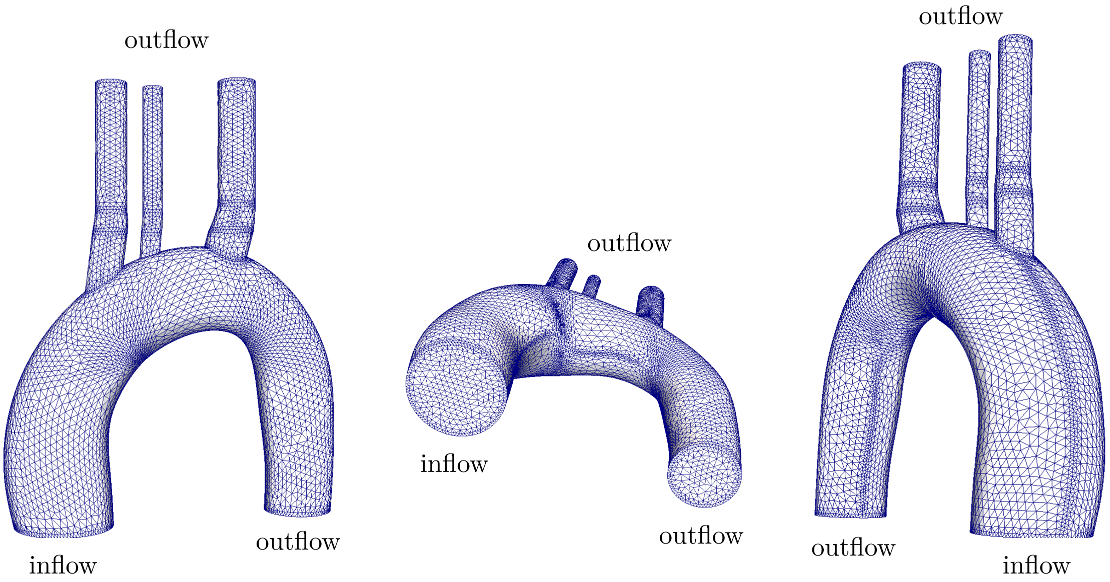


Figure 86: Aortic geometry and fluid mesh, different perspectives. The structural mesh is a simple extrusion of the outer boundary of the fluid mesh.

fluid density	ρ_F	$1.06 \times 10^3\,\mathrm{kg/m^3}$
dynamic viscosity	μ	$3.5 \times 10^{-3}\,\mathrm{kg/(m\,s)}$
reference velocity	u_0	$1.0\,\mathrm{m/s}$
structure density	ρ_S	$1.2 \times 10^3\,\mathrm{kg/m^3}$
Young's modulus	E	$4.5 \times 10^6\,\mathrm{N/m^2}$
Poisson ratio	ν	0.45
timestep size	Δt	$1 \times 10^{-3}\,\mathrm{s}$

Table 28: Physical and numerical parameters for the Aorta showcase.

Testcase Description The testcase is provided by Martorell et al. In [138], they carry out a comparative study between a healthy patient and one with an ascending aorta aneurysm. The showcase of this section only considers the healthy patient. The geometry of the aortic arch is reconstructed from four-dimensional magnetic resonance imaging. Figure 86 depicts the geometry and the fluid mesh from different perspectives. One inflow and four outflows are described. All other boundaries belong to the coupling interface. The diameter of the inflow measures approximately $2.6 \times 10^{-2}\,\mathrm{m}$. I consider surrogate boundary conditions as the actual boundary conditions from [138] are not provided by the authors. The focus of this section is not to study physical results, but to check the numerical behavior of the coupled simulation. I use a periodic inflow condition,

$$u_{in} = u_0 \cdot (1 - \cos(2\pi \frac{t}{T}))\ ,$$

with the period $T = 2.0 \times 10^{-2}\,\mathrm{s}$ and simple vanishing Neumann boundary conditions for the velocity at the outflow. The period T is chosen an order of magnitude smaller than in a realistic setting to require less simulation time for an actual moving geometry with several periods. The initial flow field is at rest. Table 28 lists the physical parameters of the testcase. The maximum inflow velocity u_0 slightly underestimates a realistic value. For a higher value, however, preliminary tests show that also realistic outflow conditions are necessary to not overestimate the resulting deformation. The outflow conditions have seven layers of 10 times higher viscosity to prevent reverse flow, compare [138]. The density ratio being close to one and the high elasticity of the structure leads to a significant added-mass effect. Together with the complex geometry, the showcase represents a very challenging case for the coupling schemes. The fluid mesh consists of 219,770 elements. The structural mesh is simply an extrusion of the boundary fluid mesh in normal direction with three layers of elements, except for the smaller branches where only two layers are used. This results in 76,883 structural elements. The extrusion leads, in particular, to a matching interface mesh with 8.183 vertices at each side. I, thus, apply a simple nearest neighbor mapping. Figure 87 shows physical results during one period after the initial phase.

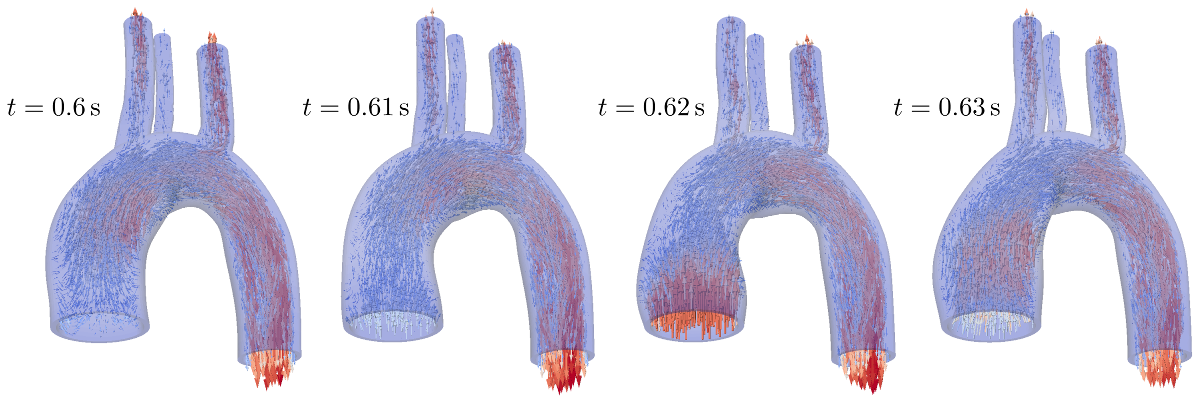


Figure 87: Aorta showcase: structural deformation and velocity glyphs for four instances during one inflow period. For visibility, the deformation is scaled by a factor of five.

Load Balancing To get a preliminary impression of the performance, I first study the strong scalability of both solvers independently in single-physics setups. To achieve a non-trivial structural setup, an (arbitrary) external volume force of 10.0 N is applied – in the negative normal direction to the inflow surface. Both solvers use a relative convergence criterion of 10^{-5} for their internal non-linear iterations and a corresponding criterion for the inner linear solvers. All tests were performed on the Haswell partition of SuperMUC, phase two, please compare the hardware description in the introduction to Chapter 4. Figure 88 depicts the strong scaling for the first 20 timesteps, neglecting the initialization. It is clearly visible that a perfect loadbalancing between both solvers is only possible if the structure solver is not run at the scalability limit. For example, 440 fluid cores and 28 structure cores looks like a balanced setup, and therefore well-suited for parallel coupling schemes. For serial coupling schemes, on the other hand, scaling the structure solver further should still result in an overall speed-up. For the coupled tests below, I use various setups of resources to study this effect.

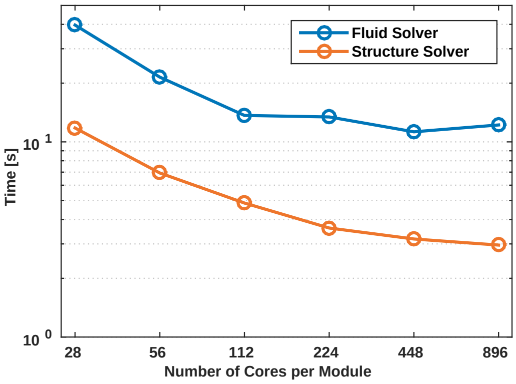


Figure 88: Aorta showcase: strong scalability of the single-physics setups and the first 20 timesteps, neglecting the initialization.

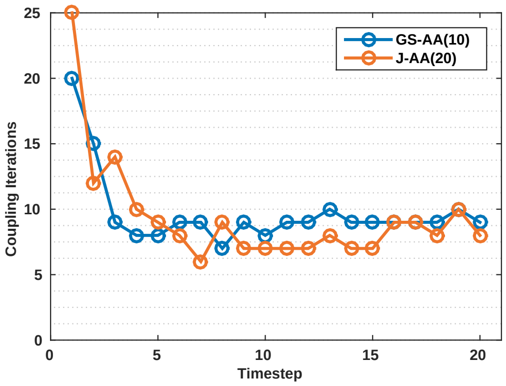


Figure 89: Aorta showcase: evolution of the coupling iterations over the first 20 timesteps for 420 fluid cores and 140 structural cores.

Performance of Various Coupling Schemes To study the performance differences between serial and parallel coupling, I apply Anderson's acceleration for both, namely GS-AA and J-AA. For comparison, I also test a constant underrelaxation of $\omega = 0.01$ and a dynamic Aitken underrelaxation. An underrelaxation with $\omega = 0.02$ already diverges. For the sake of brevity, I do not test the generalized Broyden schemes. I argue in Chapter 3 that they show a similar performance as the Anderson acceleration schemes and the focus of this section is on the comparison of serial and parallel coupling. For the reuse of columns from previous timesteps, I use the suggestions from Chapter 3: 10 timesteps for GS-AA and 20 timesteps for J-AA. A QR1 filter with $\epsilon = 10^{-5}$ is applied. The J-AA scheme uses the residual-sum weighting as preconditioner. As coupling convergence criterion, I use a relative criterion of 10^{-3} for both, displacements and forces. Finally, I use TCP/IP sockets for the communication between both solvers. For all coupling schemes, the performance after 20 timesteps is measured.

Table 29 lists the runtime and the total number of coupling iterations for the various coupling schemes and different numbers of structure cores. Aitken and constant underrelaxation, I only compare at the best sequential setup – both solvers at the scalability limit, meaning 420/140 fluid/structure cores. An immediate conclusion is the superiority of the quasi-Newton schemes compared to the underrelaxation schemes. J-AA outperforms the Aitken underrelexation in runtime by a factor of more than six, and the constant underrelexation by a factor of more than 25, for 420/140 fluid/structure cores. Both quasi-Newton schemes need roughly a similar amount of coupling iterations, which is consistent with the results of Chapter 3. GS-AA shows, however, a 17% slower runtime than J-AA for 420/140 cores Furthermore, J-AA can use less resources for the structure solver without interfering with the runtime, since the fluid solver is apparently still the slower part. Only for 420/28 cores, the structure solver becomes the bottleneck. The superiority of J-AA holds despite the fact that the quasi-Newton system itself becomes significantly more expensive to solve – roughly by a factor of 12. For completeness, Figure 89 shows the evolution of the number of coupling iterations during the first 20 timesteps. Similar to the results of Chapter 3, J-AA shows a worse starting behavior than GS-AA, until a significant column space is constructed.

To summarize, the parallel coupling scheme J-AA results in a significant speed-up compared to the serial coupling scheme GS-AA, while using less resources. Please note that, in principle, GS-AA would allow to reuse resources for both solvers, which would lead to a reduction of the total resource to 420 cores. A realization is, however, technically cumbersome as memory would have to be shared among both solvers and the communication between both solvers would have to be realized in a way that does not block resources. Finally, GS-AA would then still show a worse overall simulation time as J-AA.

	F/S cores	GS-AA(10)	J-AA(20)	GS-Aitken	GS-const.(0.01)
Total Iterations	420/28	189	189	-	-
Iterations per Timestep	420/28	9.45	9.45	-	-
Total Runtime [s]	420/28	**244.5**	**168.9**	-	-
QN Runtime [s]	420/28	0.67	10.11	-	-
Total Iterations	420/56	187	188	-	-
Iterations per Timestep	420/56	9.35	9.4	-	-
Total Runtime [s]	420/56	**185.1**	**137.0**	-	-
QN Runtime [s]	420/56	0.67	11.29	-	-
Total Iterations	420/140	194	189	1194	8803
Iterations per Timestep	420/140	9.7	9.45	59.7	440.15
Total Runtime [s]	420/140	**160.5**	**137.7**	902.0	3477.0
QN Runtime [s]	420/140	0.81	10.21	-	-

Table 29: Aorta showcase: total runtime and number of iterations for the first 20 timesteps and for various coupling schemes and compute resources. Runtimes are averaged values over five simulations, neglecting the fastest and the slowest experiment. Aitken and constant underrelaxation is only tested for 420 fluid and 140 structure cores.

5.3 Fluid-Structure-Acoustics Interaction

The ExaFSA project[47], part of the German priority program SPPEXA[48], exemplarily studies fluid-structure-acoustics interaction (FSAI)[130, 166, 175] as a challenging three-field coupled problem and its efficient realization on modern compute hardware. FSAI plays an important role for noise-reducing design of technical devices such as aircrafts, fans, or wind turbines. FSAI is, in particular, challenging as it features multi-scale properties besides the multi-physics nature. The showcases of this section present the intermediate results of the ExaFSA project after its first of two phases. I focus, however, solely on a technical point of view and not on physical results, meaning the technical realization of the three-field coupling, the inter-solver parallelism and inter-solver load balancing. In a way, this section can be regarded as a generalization of the two-field inter-solver parallelism of Chapter 3. I start the showcase with a brief introduction to the coupled problem of FSAI in Section 5.3.1. As FSI is already widely discussed in this thesis, I focus on fluid-acoustics interaction (FAI). If the flow field can be decomposed into a near-field with fully resolved flow and an acoustic far-field, FAI becomes a surface-coupled two-field problem. It is obvious that FAI is an important milestone towards full FSAI. Afterwards, in Section 5.3.2, I study the FAI of a 2D subsonic jet. For this scenario, the near-field is further decomposed into an inner viscid and an outer inviscid domain. Thus, for this application, FAI already becomes a three-field coupled problem. I exemplarily study the load-balancing between all three solvers for this scenario. Finally, Section 5.3.3 studies the full FSAI of the flow around a bending tower. As I, therefore, mainly summarize our work in [25], I keep this last section short.

5.3.1 Fluid-Acoustic Interaction

When looking at the noise emission of a wind turbine, the multi-scale nature of FSAI becomes obvious: noise is generated in the boundary layer of the wind turbine at a length scale of centimeters, the whole turbine has a length scale of meters, and important noise immission zones might be in a distance of hundreds of meters. Thus, resolving all scales of such a scenario with a direct monolithic simulation is computationally unfeasible. On the other hand, acoustics is nothing else than travelling pressure waves in fluids. Close to the wind turbine, theses waves superpose pure flow phenomena (cf. e.g. [238]). Further away from the turbine, however, fluid phenomena can be neglected. Thus, splitting of the fluid domain into near-field and far-field is possible. The near-field marks the domain where acoustic waves are generated, possibly as part of an FSI. Here, a full compressible flow simulation is necessary. In the far-field, only acoustic waves need to be resolved, whereas the flow itself can be neglected. Linearized equations around a fixed background state can be applied. Contrary to FSI, the location of the splitting is, however, not obvious. Figure 90 depicts the three-field setup of FSAI and a suitable coupling strategy. On the left, the three zones, structure, fluid, and acoustics, are depicted. Between fluid and structure, typically, a bi-directional coupling is necessary. Forces and displacements need to be exchanged, for example, as explained in Section 3.1. The coupling at the FAI interface is different: If we assume that the far field contains no obstacles that might reflect acoustic waves back into the near field, the coupling can be modeled as uni-directional. Furthermore, the complete state, velocity, pressure, and density, needs to be communicated. These two different kinds of interaction also influence the coupling strategy as depicted on the right side of Figure 90. As the interaction between fluid and structure might be strong, an implicit coupling is necessary. For the FAI coupling, however, an explicit coupling is sufficient due to the uni-directional dependence. At the same time, acoustic phenomena occur also on a different time-scale than flow phenomena. A sub-cycling of the acoustic solver might be desirable.

The partitioning into near-field and far-field has the additional advantage that tailored numerical schemes and resolutions can be applied in every sub-domain. In the near field, a fine mesh resolution is necessary to capture the geometry movement of the FSI. Thus, a low order scheme is sufficient. Typically, a finite volume discretization is a good choice for the near-field. In the far-field, however, a coarse resolution is sufficient as geometries do not have to be resolved carefully. Thus, a higher-order scheme can be applied. As the far-field is a purely linear problem, a higher-order scheme does also not become overly expensive. Furthermore, the lower numerical dissipation fits well to the wave phenomena. Thus, discontinuous Galerkin schemes, which became popular during the last decade, are a very well-suited choice for the far-field.

[47] https://ipvs.informatik.uni-stuttgart.de/SGS/EXAFSA/

[48] http://www.sppexa.de/

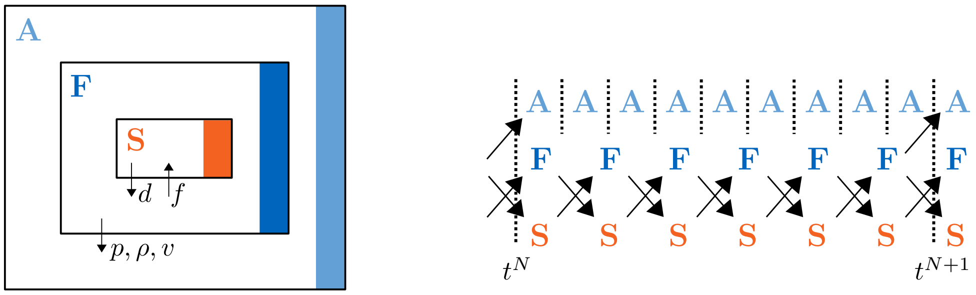


Figure 90: Schematic view on fluid-structure-acoustic interaction, domain decomposition (left) and coupling scheme (right). d: displacements, f: forces, p: pressure, ρ: density, v: velocity. Arrows mark data exchange. Doted lines mark timesteps. t^N, t^{N+1} denote global timesteps.

With this motivation, I end the brief introduction into FAI and FSAI. For a more detailed description, the reader may refer to our work in [25].

5.3.2 Three-Field Flow Coupling around a 2D Subsonic Free Jet

As an important milestone towards full FSAI, this section discusses the coupling strategy and the performance results of a three-field flow coupling around a 2D subsonic free jet. Directly around the development of the jet, the compressible Navier-Stokes equations are used to fully resolve the flow field. Further away – in the mid-field – viscous forces can be neglected, such that the computationally cheaper Euler equations can be used. In the far-field, only acoustic waves are resolved by means of Euler equations that are linearized around a fixed background state. At both coupling interfaces, the complete state variables – density, velocity, and pressure – are exchanged bi-directionally. From the Euler domain to the Navier-Stokes domain, also the gradients of the state variables are exchanged. To allow for a faster reading, I abbreviate the three coupled solvers as NS, E, and LE in the following.

Testcase Description Figure 91 depicts the geometrical layout of the Jet scenario and Table 30 lists the physical parameters of the testcase. Initially, the flow is at rest, identical to the background state.

background fluid density	ρ^0	$1.4\,\mathrm{kg/m^3}$
background fluid velocity	u^0	$(0.0, 0.0)$ m/s
background fluid pressure	p^0	$1.0\,\mathrm{N/m^2}$
specific gas constant	R	$2.8 \times 10^2\,\mathrm{m^2/(s^2\,K)}$
thermal conductivity	λ	$1.4 \times 10^{-4}\,\mathrm{kg\,m/(s^3\,K)}$
dynamic viscosity	μ	$1.0 \times 10^{-7}\,\mathrm{kg/(m\,s)}$
isentropic coefficient	γ	1.4
speed of sound	c	$1.0\,\mathrm{m/s}$
jet radius	r_0	$1.0 \times 10^{-1}\,\mathrm{m}$
momentum thickness	d	$5.0 \times 10^{-3}\,\mathrm{m}$
timestep size	Δt	$5 \times 10^{-5}\,\mathrm{s}$

Table 30: Physical parameters of the Jet scenario.

In the NS domain, the velocity in x-direction at the inflow is described as

$$u_x = \tilde{u}_x \cdot 0.5 \cdot \left(1 + \tanh\left(\frac{r_0 - |y - y_0|}{2d}\right)\right) ,$$

with $\tilde{u}_x = Ma \cdot c$ and the Mach number $Ma = 0.4$. The tanh function is used to smooth the pulse in y-direction. The jet's center is slightly moved in y-direction by $y_0 = 1 \times 10^{-4}\,\mathrm{m}$ to induce asymmetry and, thus, foster the development of vortices in the flow. The velocity in y-direction is set to zero. To diminish the initial shock of the jet streaming into a fluid at rest, the inflow velocity is ramped up to the full amplitude by means of sinusoidal factor during the first 10 s. Since no actual jet nozzle is resolved

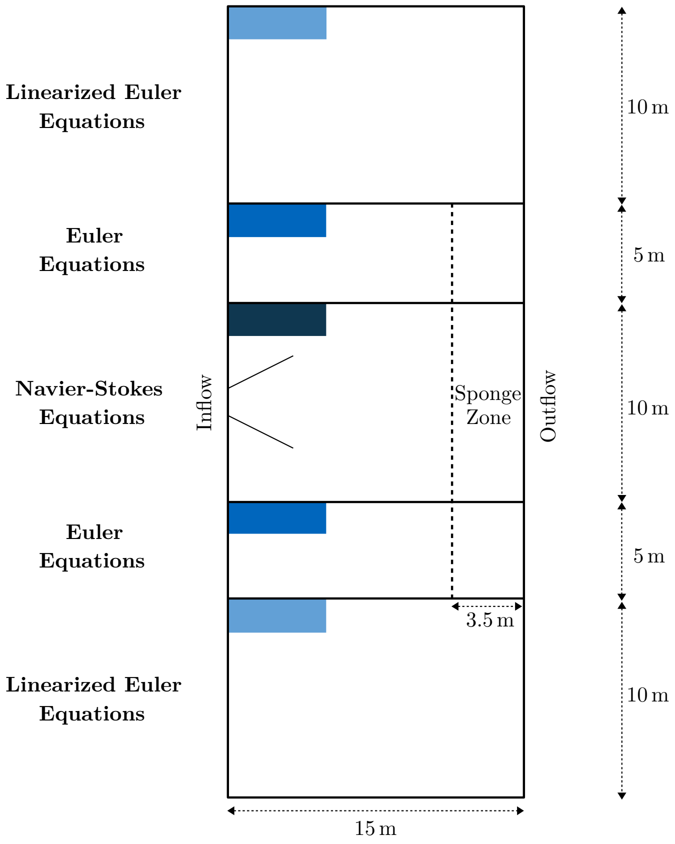


Figure 91: Geometrical layout of the Jet testcase. In the Navier-Stokes domain, a free subsonic jet is described as inflow condition. Further away from the jet, first, viscous forces can be neglected (Euler domain), and afterwards, complete flow phenomena (linearized Euler domain). A bi-directional coupling is used at all coupling interfaces.

explicitly, the density at the inflow is adapted to the velocity by means of the Crocco-Busemann relation (cf. [17]),

$$\rho = \tilde{\rho} \cdot \left(1 + 0.5 \cdot (\gamma - 1) \cdot Ma^2 \cdot \frac{u_x}{\tilde{u}_x} \cdot \left(1 - \frac{u_x}{\tilde{u}_x}\right)\right)^{-1} ,$$

with $\tilde{\rho} = 2.0\,\mathrm{kg/m^3}$. The pressure at the inflow follows a Neumann-zero condition. The inflow boundary outside the jet radius in the NS and the E domain is fixed to the background state.

At the outflow, in the NS and in the E domain, the pressure is fixed to the background state while the velocity and the density follow a Neumann-zero condition. Furthermore, a sponge zone is used to artificially damp the flow to the background state and, thus, to allow flow features to leave the domain without reflections. Finally, all outer boundaries in the LE domain fix the acoustic pertubations to zero.

Numerical Settings All three domains use the discontinuous Galerkin solver Ateles, compare Section 2.4. The spatial order is set to 16 and the grid resolution to 0.25 m. Thus, both the NS domain and the E domain include 2,400, and the LE domain 4,800 elements. Please note that such an identical spatial order in every domain is not optimal, but merely chosen for the sake of simplicity. A better choice would be a finer mesh with a lower order in the NS domain, a coarser mesh with a higher order in the LE domain, and the E domain somewhere in between. In particular, with the current setup, all coupling interfaces feature matching meshes. Still, preCICE exchanges both meshes to also allow for consistent mappings in both direction in case of non-matching meshes at a later point in time – please recall the discussion in Section 4.3.1. All domains use a second order explicit Runge-Kutta scheme for the time integration. Due to different stability restrictions, however, each domain would require a different maximum timestep size. For the current setting, in the NS domain, the timestep size would have to be an order of magnitude smaller than in the E and in the LE domain. This is, indeed, another important advantage of the splitting in NS and E. For the sake of simplicity, however, the overall timestep size is fixed to the smallest restriction – here the NS criterion, leaving room for improvement (compare the discussion in Section 6.2). For the coupling, two parallel-explicit coupling schemes are composed, please recall Section 3.8.2. Here, the coupling between NS and E precedes the coupling between E and LE. Still, all three solvers are executed in parallel to each other, as visualized in Figure 92.

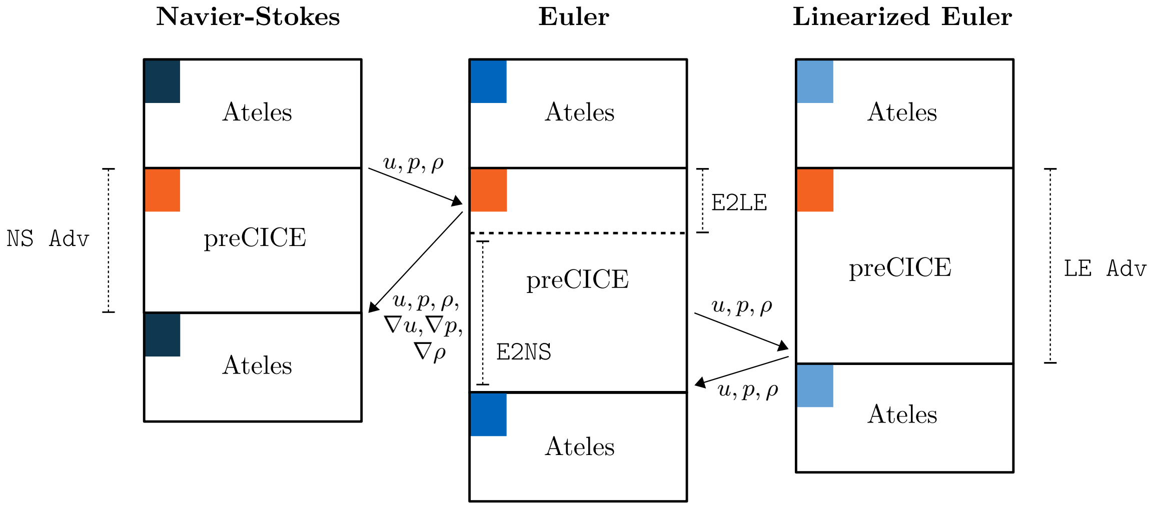


Figure 92: Jet testcase, flow chart of the three-field coupling. Temporal flow is from top to bottom. The combination of two explicit-parallel coupling schemes allows the simultaneous execution of all three solvers. The Euler solver first exchanges data with the Navier-Stokes solver and afterwards, with the linearized Euler solver. The time events are: `NS Adv` – the `advance` call of the Navier-Stokes solver, `LE Adv` – the `advance` call of the linearied Euler solver, `E2NS` – the coupling between the Euler and the Navier-Stokes solver, and `E2LE` – the coupling between the Euler and the linearized Euler solver. The sum of the latter two gives the time spent in the `advance` call of the Euler solver – `E Adv`. Please note that `E2NS` already ends after the asychronous send from the Euler solver.

Figure 93 shows the pressure in all three domains at various points in time during the development of the jet. A smooth transition over all interfaces is achieved, confirming the correct coupling and justifying the position of the interfaces. The superposition of flow phenomena and acoustic waves is clearly visible. Eventually, the simulation crashes at $t = 133\,\mathrm{s}$, when the first vortex is about pass from the E to the LE domain, contradicting the linearization.

Performance and Load Balancing I use the Jet showcase to exemplarily study the performance and load balancing of a three-field coupled problem. As the initialization cost for coupled scenarios with Ateles are already thoroughly discussed for the Ateles Cube in Section 4.5.1, I focus solely on the cost per timestep. The load balancing of a three-field coupling is non-trivial to analyze. NS is expected to be slightly more expensive than E, since the equations involve more terms. LE, on the other hand, should again be much cheaper than E, since the latter requires Fourier transformations back and forth between the modal and the nodal space per timestep. Measurements showed that this lets the cost of E scale quadratically with the spatial order, while the cost for LE only scales linearly. The LE domain is, furthermore, double the size than each one of the other two. The basic idea to analyze the three-field load balancing is to run either NS or LE with a very high amount of resources. Then, the remaining two solvers can be balanced out. The balance itself can be implicitely studied by considering the time spent in the preCICE function `advance`, as it also contains a possible synchronization time between the various solvers. Furthermore, I split up the time spent in the `advance` call of E into both coupling schemes. In total, I consider the five events `NS Adv`, `E Adv`, `LE Adv`, `E2NS`, and `E2LE`, as also depicted in Figure 92. For the analysis of the events, two effects have to be taken special care of. First, the time needed for the load balancing always overlays with the actual communication, which also scales with the number of cores. The mapping and any other computations in preCICE, on the other hand, are negligible for this showcase. Second, the end of the event `E2NS` is already triggered once the asynchronous sending

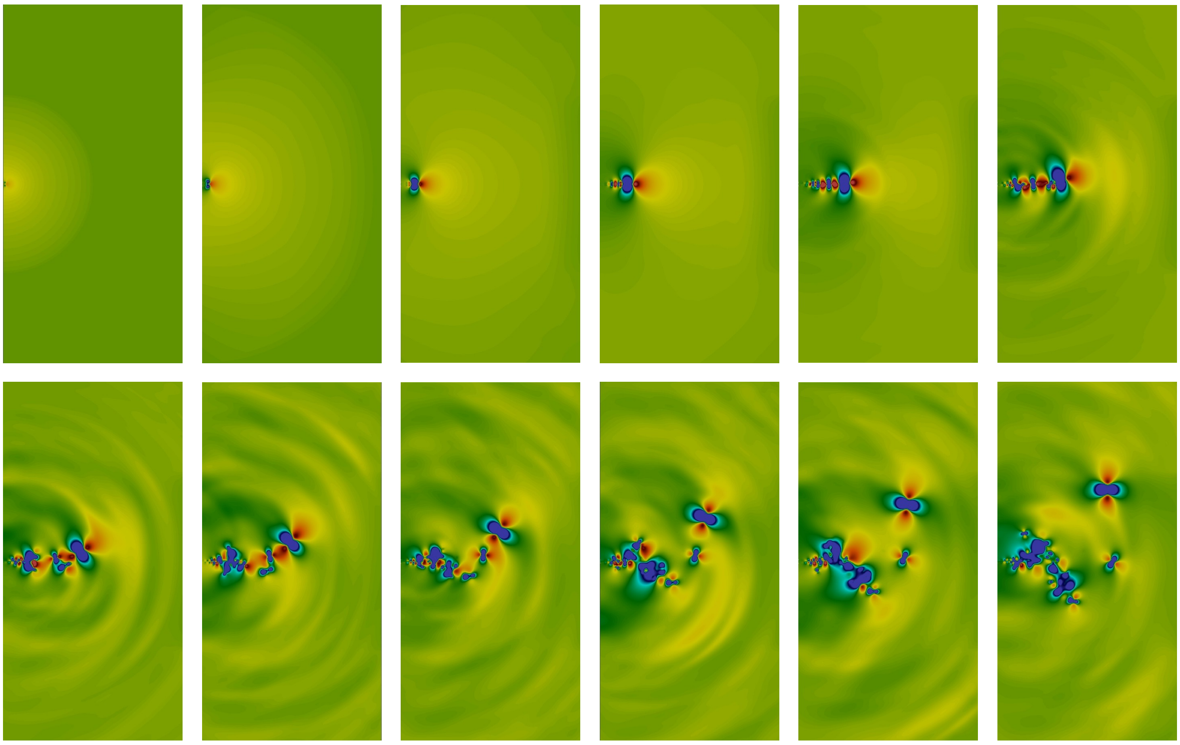


Figure 93: Acoustic pressure waves through all three domains of the Jet testcase. From left to right and from top to bottom, at $t = 10.0\,\mathrm{s}, t = 20.0\,\mathrm{s}, \dots, t = 120.0\,\mathrm{s}$. The color scale ranges from $0.974\,\mathrm{N/m^2}$ to $1.027\,\mathrm{N/m^2}$

operations from E to NS are finished. The communication itself is, however, not yet completed and, thus, contributes artificially to the time spent in `E2LE`. This phenomenon is also indicated in Figure 92.

I run all tests on the thin nodes partition of SuperMUC, phase one, please recall the hardware description in the introduction to Chapter 4. As usual, I perform five runs for each test, discarding the minimum and the maximum, and averaging the remaining three runtimes. For the M2N communication, I use TCP/IP sockets, which show an approximately seven times worse performance than MPI Ports in Section 4.2.3. Still, the MPI Ports show ever consisting robustness issues on SuperMUC, such that tests after the SuperMUC software update of spring 2016 did no longer work, compare the discussion in Section 4.2.3. I measure the time after the first 1,000 timesteps.

For the first series of experiments, the NS domain uses 40 nodes, the LE domain one node and the E domain varies between 4 and 18 nodes. For this setup, NS should always be the fastest solver such that the other two can be balanced out. Figure 94 shows the timings of all events over the varying E domain size. For lower E resources, LE always has to wait for E as visible in the high timing values of `LE Adv`. For more E resources, E becomes faster than LE resulting in low `LE Adv` values. `E2LE` first decreases due to faster and faster parallel communication of E. Afterwards, after 10 Euler nodes, `E2LE` increases due to the increasing waiting time of E. Please note that this increase in waiting time looks, at first, smaller than the decrease of waiting in `Adv LE`, but both changes have to be compared in a relative manner. Next, `E2NS` is relatively small, since, before the receive of E, NS has always already sent its data. Furthermore, the sending of data from E to NS is not measurable in `E2NS` as explained in Figure 92. `Adv NS` decreases as long as E becomes faster. Afterwards, LE is the bottleneck, which remains constant. I conclude that the optimal load balancing between LE and E is approximately 1 to 10 for the complete scenario, meaning 1 to 20 per element.

For the second series of experiments, E is fixed at 10 nodes, LE at 5 nodes, and NS varies between 10 and 24 nodes. The results above indicate that LE should now always be the fastest solver, which allows to balance out NS and E. Figure 95 visualizes the results. The situation is slightly more complicated to interpret. Jumps from 16 to 18, and from 22 to 24 NS nodes are noticeable. They are assumably due to the higher amount of nodes of NS that lie at the coupling interface. They let the communication time

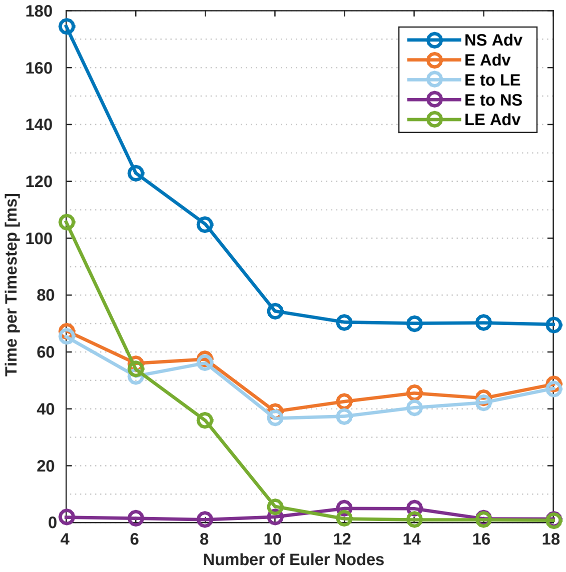


Figure 94: Load balancing study for the Jet testcase. The Navier-Stokes solver uses 40 nodes and the linearized Euler solver one node, while the resources for the Euler solver are varied.

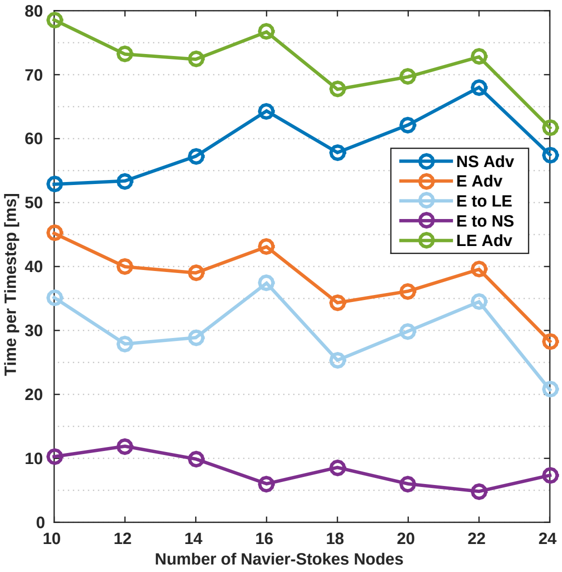


Figure 95: Load balancing study for the Jet testcase. The Euler solver uses 40 nodes and the linearized Euler solver five node, while the resources for the Navier-Stokes solver are varied.

for NS decrease due to higher parallelism, but increase the communication time for E, since messages to more receivers have to be sent. If we subtract this superposed effect, the waiting time for NS is constant from 10 to 12 nodes and increases afterwards. Consistently, `E2NS` increases from 10 to 12 nodes and decreases afterwards. Thus, I conclude that the optimal load balancing between NS and E is approximately 12 or 14 to 10. For the sake of completeness: `E2LE` needs a constant time for the actual communication, but is superposed by the sending from E to NS, and `Adv LE` shows the expected high waiting time including the effects of the slower one of either E or NS. Please note, that the fastest overall setting, judging by `Adv LE` at 24 nodes for NS, is not the best load balancing. If the overall amount of resources is not conserved, best load balancing does, of course, not imply fastest overall setup.

Finally, I test how the load balancing and the general performance changes when the approximately ideal balance 14 to 10 to 1 is scaled up. I start with 14 nodes for NS, 10 nodes for E, and one node for LE, and double the amount of nodes three times. Figure 96 shows the timing of all events, but also the time solely spent in Ateles, exemplarily for E. The load balancing changes when scaling up: LE is, at first, the overall bottleneck, but becomes faster than the other two solvers for more than two nodes. The balance between NS and E slightly moves towards a relatively faster NS for a higher amount of resources. In general, preCICE cannot keep up with the scaling of Ateles, since the interface sizes do not scale perfectly. Table 31 shows that number of cores at the interfaces does not double when the overall number of cores doubles. One should also not expect this: for a perfectly uniform refinement in 2D, the cores at the coupling interface should increase by a factor of $\sqrt{2}$ when the overall number of cores is doubled. This is worse than in 3D[49] and explains the discrepancy to the Ateles Cube results of Section 4.5.1 to some extent. Table 31 further lists the maximum amount of interface vertices per core (IVpC), which further illustrates this phenomenon. Furthermore, the non-scaling communication overhead has to be taken into account. The mediocre performance lets preCICE easily become more expensive than Ateles. If MPI communication could be used instead of TCP/IP, the communication is, however, expected to drop by a factor of seven, as already mentioned further above, which would change the overall picture drastically.

[49] If the domain decomposition in 3D refines along one axis per doubling of the resources, three such refinement steps lets the amount of cores at an axis-aligned plain double twice. This gives a factor of $\sqrt[3]{4} \approx 1.59 > 1.41 \approx \sqrt{2}$.

Cores NS	224	448	896	1972
Cores E	160	320	640	1280
Cores LE	16	32	64	128
Int. Cores NS	43	55	77	91
Int. Cores E2NS	36	43	72	72
Int. Cores E2LE	32	57	61	62
Int. Cores LE	6	10	14	22
Max. IVpC. NS	80	64	32	32
Max. IVpC. E2NS	80	64	48	32
Max. IVpC. E2LE	80	64	48	32
Max. IVpC. LE	496	368	208	160

Table 31: Changing interface resources for the strong scalability test of the Jet testcase. The overall number of cores in each domain is compared to the number of cores at the coupling interfaces. Furthermore the maximum number of interface vertices per core (IVpC) is listed to judge upon the load balancing within each interface.

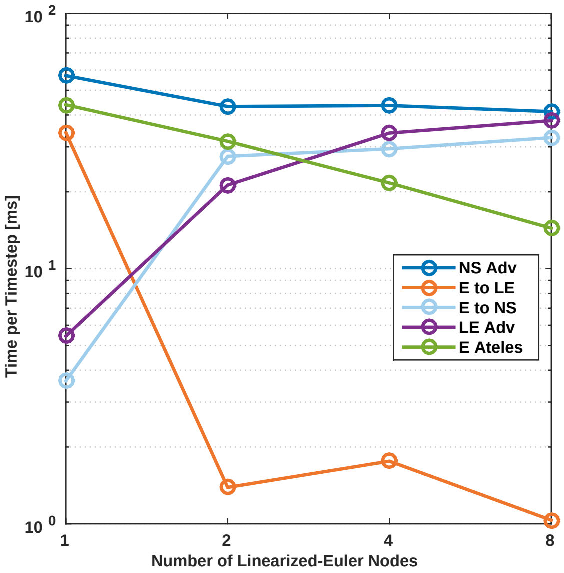


Figure 96: Strong scalability test for the Jet testcase. The nodes of the Navier-Stokes, the Euler, and the linearized Euler solver use a fixed ratio of 14 to 10 to 1, and are doubled three times.

5.3.3 Fluid-Structure Acoustic Coupling for a 3D Bending Tower

As final example of this showcase section, I give intermediate results for a full FSAI example, a 3D bending tower in cross flow [25, 26]. The scenario is rather simple, but includes three-dimensional effects. Figure 97 depicts the geometry of the scenario. The fluid domain uses compressible Navier-Stokes equations and OpenFOAM as solver. The structural domain also uses OpenFOAM. Similar to the Jet scenario, the acoustics domain again uses linearized Euler equations and Ateles as solver. Between fluid and structure, a quasi-Newton implicit coupling scheme is used while the fluid-acoustic coupling is explicit and uni-directional, similar as depicted in Figure 90.

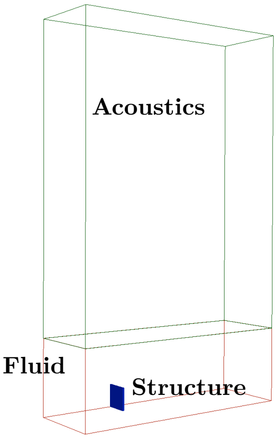


Figure 97: FSAI showcase, 3D Bending Tower: geometry setup. The figure is adapted from [25].

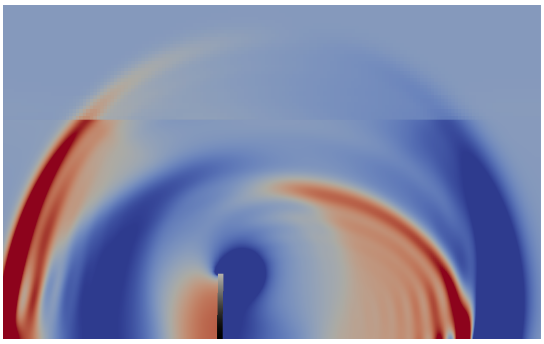


Figure 98: FSAI showcase, 3D Bending Tower: acoustic pressure waves and structural displacement. The figure is adapted from [25].

The flexible structure is fixed at the no-slip bottom. At the left and the right boundary, periodic conditions are prescribed. In span-wise direction, symmetric boundary conditions are used. The acoustic solver sets the acoustic pertubation to zero at the top boundary, Dirichlet coupling conditions at the bottom, and periodic conditions at all other boundaries. Table 32 lists all parameters of the scenario.

Initially, the flow is set to the non-zero background state. The structure uses 4,500 and the fluid approximately 300,000 control volumes. The acoustics domain includes 6,144 elements with a spatial discretization order of seven. Figure 98 shows acoustic pressure waves – results borrowed from our work in [25]. The transition through the coupling interface features a visible jump, which we assume to be due to the outflow condition that the fluid solver uses at the top boundary or the different boundary conditions in span-wise direction. Still, the results proof that preCICE is able to technically handle the implicit-explicit three-field coupling.

background fluid density	ρ_F^0	$1.0\,\mathrm{kg/m^3}$
background fluid velocity	u^0	$(2.3, 0.0, 0.0)$ m/s
background fluid pressure	p^0	$1.0 \times 10^2\,\mathrm{N/m^2}$
kinematic viscosity	ν_F	$1.0 \times 10^{-2}\,\mathrm{m^2/s}$
speed of sound	c	$11.8\,\mathrm{m/s}$
structure density	ρ_S	$1 \times 10^3\,\mathrm{kg/m^3}$
Young's modulus	E	$1.4 \times 10^6\,\mathrm{N/m^2}$
Poisson ratio	ν	0.4
timestep size	Δt	$1 \times 10^{-5}\,\mathrm{s}$

Table 32: Physical parameters of the Bending Tower scenario.

5.4 Turbulent Flow around an Elastic Hemisphere

In [227], the turbulent flow around a rigid hemisphere is studied both experimentally and via simulation. Air inflated hemispheres play a role in modern civil engineering, for example, as buildings for temporary housing in disaster areas. To study the stability of such constructions, a full FSI simulation is, however, mandatory, as also mentioned in [227].

To realize an FSI simulation, the immense cost asymmetry of the scenario needs to be taken care of. On the one hand, the fluid simulation needs a careful resolution of the complete boundary layer, especially due to the spherical shape, leading to an expensive setup with a very fine resolution of the coupling interface. The structure simulation, on the other hand, might only feature a few hundreds of degrees of freedom as a sphere can be efficiently modeled by shell elements. I already discuss the consequences of this asymmetry in Section 1.1.2: the FSI approach needs to take advantage of the asymmetry, meaning that only the small mesh should be communicated. In Section 4.5.2, I show a performance study for the asymmetric Pfs-1a benchmark. The even more expensive hemisphere setup takes this effect to extremes: an server-based FSI approach appears unfeasible. Therefore, the case marks a perfect showcase for the intra-solver parallelism of preCICE.

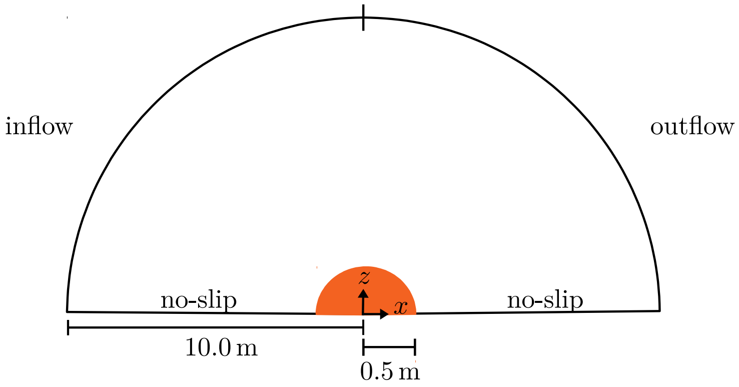


Figure 99: Geometrical layout of the Hemisphere scenario, cut through the $y = 0$ plain. The elastic inner structure is depicted in orange. The sketch is not true to scale.

For the FSI simulation, I use Alya Nastin as fluid solver and reuse the same setup as in [227], including the mesh, which the authors provided. The mesh consists of approximately 31 million nodes, of which 128,000 lie at the coupling interface. As structure solver, I use Carat++. The hemisphere is modelled with only 228 nodes. Figure 99 shows the geometry of the scenario. The complete fluid domain is also a hemisphere with a 20 times larger diameter than the elastic hemisphere, which is placed in the center of the domain. The outer boundary is split into an inflow boundary and an outflow boundary. The bottom as well as the structure use a no-slip description. The inflow condition uses a turbulent wall profile of the height of the elastic hemisphere and a constant profile above. As the Alya configuration currently does not support a `min` function to concatenate both profiles, I approximate the profiles via

$$\arctan(10z) \cdot \frac{2}{\pi} \cdot 0.862351 \cdot u_0 \ .$$

Table 33 lists all parameters of the scenario.

Results A technical MPI issue on SuperMUC, which appeared after the system update of Spring 2016, could not be resolved in time for this thesis. Therefore, unfortunately, I have to fall back to a single feasibility run that I ran before the system update. A thorough study and a joint publication with all involved researchers is, however, planned for the near future. The single run uses 1.764 cores for the fluid problem and a single core for the structure problem, both on the Haswell partition of SuperMUC. The FSI simulation uses a stabilized fluid run as initial condition. For the coupling, I apply a parallel Anderson acceleration with reused columns from 20 timesteps, J-AA(20), together with the residual-sum weighting. The coupling iteration uses a relative convergence criterion for both displacements and forces of 10^{-3}. Both solvers use a relative criterion of 10^{-5} for their internal non-linear solver. The

fluid density	ρ_F	$1.225\,\mathrm{kg/m^3}$
dynamic viscosity	μ	$1.826\,475 \times 10^{-5}\,\mathrm{kg/(m\,s)}$
reference velocity	u_0	$1.0\,\mathrm{m/s}$
structure density	ρ_S	$1.7 \times 10^{3}\,\mathrm{kg/m^3}$
Young's modulus	E	$1 \times 10^{5}\,\mathrm{N/m^2}$
Poisson ratio	ν	0.3
internal pressure	p_0	$1.0\,\mathrm{N/m^2}$
timestep size	Δt	$1 \times 10^{-2}\,\mathrm{s}$

Table 33: Physical and numerical parameters for the Hemisphere showcase.

first 58 timesteps result in an average of 3.26 coupling iterations per timestep, which shows that the Anderson acceleration results in a very efficient coupling. In particular, the turbulent fluctuations have no negative influence on the stability of the coupling scheme. Assumably, this is due to the fact that the acceleration is computed on the coarse structure mesh, not on the fine fluid mesh. Furthermore, the fully-parallel concept of preCICE can take full advantage of asymmetry such the coupling cost is almost negligible. The initialization in preCICE, for example, only takes 0.2 s. To summarize, both the coupling schemes as well as the intra-solver parallelism in preCICE seem very well suited for the Hemisphere scenario, although a detailed study should follow.

5.5 Simulation of a Brush Seal – Feasability Study

Colleagues at the chair of Turbomachinery and Flight Propulsion, TUM, in a cooperation with MTU Aero Engines, study the behavior and design of brush seals. To approach realistic simulations of these components, simplified FSI setups are tested, both experimentally and via simulation. To this end, the vibrations of wall-mounted thin steel cylinders are studied. In [70, 172], the vibration of a single cylinder is already exemplarily simulated. The showcase of this section considers a setup with multiple cylinders, which interact with each other. I refer to this setup as Multi-Cylinder testcase. The testcase is well suited to demonstrate the outcome of this thesis. First, the inherent flexibility of the partitioned approach is obligatory, as sophisticated and well-validated fluid and structure solvers should be reused in the cooperation. Second, the multi-coupling scheme, developed in Section 3.8, can be used to couple various cylinders with one fluid solver. Third and last, turbulent flow together with the FSI leads to a computational expensive simulation for which the parallelization concepts of this thesis can be applied. Next, I briefly introduce the reader to brush seals, following the description from [172], and to the applied simplifications. Afterwards, I detail the simulation setup and discuss the results.

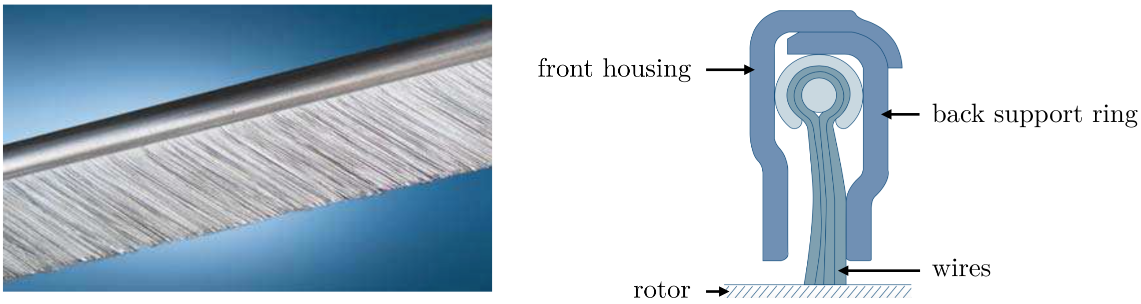


Figure 100: Sealing element of a brush seal (left) and schematic cross section of a sealing element with its housing (right). Both figures are taken from [70] and are already used in [172].

Brush Seals Brush seals are used in a variety of applications, such as flight propulsion systems, gas turbines and steam turbines. These applications have in common that an efficient sealing between static (stator) and dynamic (rotor) elements of a machine is necessary to separate gases from gases, but also gases from liquids. Here, brush seals show a significantly lower leakage than alternative, conventional technologies (cf. [70]). Figure 100 depicts such a sealing element. Obviously, the name of these seals originates from their characteristic structure: many slender and elastic wires are clamped together to form a brush. In the initial configuration, these wires are spatially quite loose. They are in contact with the rotor, but permit both axial and radial relative motion between stator and rotor. As soon as the rotor starts moving, the brush wires are pushed against the back support ring. Now, the wires align and the brush becomes a leak-tight barrier. To better understand this behavior, especially the influence of FSI, a stepwise approximation with both simulation and experimental study is carried out.

Simplified Fluid-Structure Interaction For simplification, we ignore the housing and any contact mechanics, but simply study the interaction of the wires with the flow. As already mentioned above, [70, 172] already study a single wire. As the mutual influence of several wires is assumably of importance to understand the exact behavior of a complete brush seal, we extend the study in this showcase to nine wires. To this end, we use a separate structure solver for each individual wire. In [59], a, from FSI perspective, similar setup is studied – multiple identical cylinders in turbulent flow – despite a different underlying application. Here, the authors use a single overall structure solver and connect the tips of the cylinders by thin artificial beams for stabilization. The usage of multiple structure solvers, as in our approach, allows the reuse of the complete setup including the mesh of one single wire. Furthermore, it is not clear if a general structure solver can numerically deal with such non-connected cylinders and how this influences its performance. For this showcase, we use SU2 as fluid solver and one Alya Solidz instance for each cylinder. SU2 simulates compressible flow, though the setup is in a nearly incompressible regime. Furthermore, as turbulence model, the Spalart-Allmaras model [188] is applied.

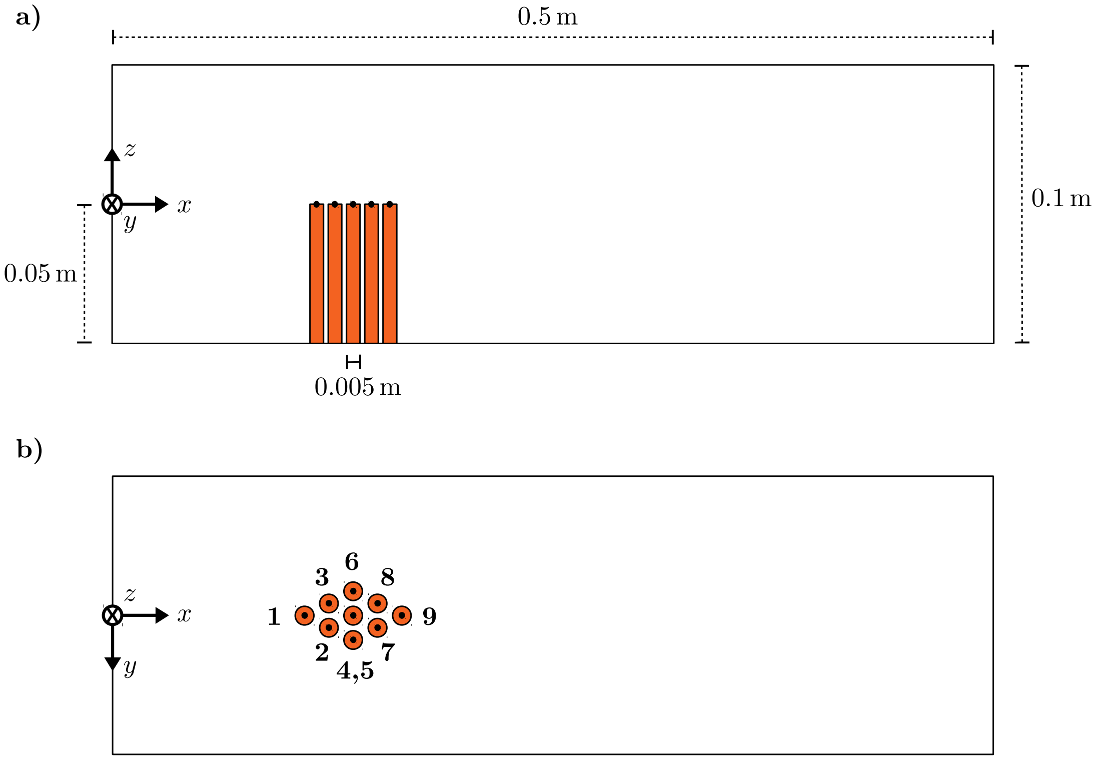


Figure 101: Schematic view of the Multi Cylinder testcase. a) cut through the plain $y = 0$, b) cut through the plain $z = 0$. The sketches are not true to scale. The exact positions of the cylinders are listed in Table 34. Similar figures are also used in [172].

	x	y	z
C_1	0.086	0.0	0.0
C_2	0.093	0.007	0.0
C_3	0.093	-0.007	0.0
C_4	0.1	0.0014	0.0
C_5	0.1	0.0	0.0
C_6	0.1	-0.0014	0.0
C_7	0.107	0.007	0.0
C_8	0.107	-0.007	0.0
C_9	0.114	0.0	0.0

Table 34: Multi-Cylinder testcase: coordinates of the top center points of all cylinders.

Scenario Description Figure 101 sketches the geometry of the Multi-Cylinder testcase. Additionally, Table 34 lists the coordinates of the top center points of all cylinders. The inlet uses a constant inlet profile, the outlet uses a standard Neumann-zero outflow condition and the bottom as well as the cylinders use a no-slip description. The top of the domain is a free stream boundary, while symmetric conditions are used in span-wise direction. A pre-computed fluid simulation is used as initial condition. To further stabilize the start of the simulation, the forces on the cylinders are linearly ramped up over the first 0.01 s. Table 35 lists all parameters of the testcase.

Numerical and Computational Setup Figure 102 visualizes the fluid mesh, which contains 417,155 control volumes. The structure solvers use between 3,883 and 3,925 elements. At all coupling interfaces, the surface meshes match. The coupling sub-iteration uses a relative convergence criterion of 10^{-3} for forces and displacements at all interfaces. SU2 uses an internal relative criterion of 10^{-7}, while all Solidz

fluid density	ρ_F	$1.185\,\mathrm{kg/m^3}$
dynamic viscosity	μ	$1.831 \times 10^{-5}\,\mathrm{kg/(m\,s)}$
specific gas constant	R	$2.870\,58 \times 10^{2}\,\mathrm{J/(kg\,K)}$
specific heat ratio	κ	1.4
Reynolds number	Re	1.0×10^{3}
Reynolds length	l_0	$5.0 \times 10^{-3}\,\mathrm{m}$
Mach number	Ma	0.1
free stream temperature	T	$2.9815 \times 10^{2}\,\mathrm{K}$
structure density	ρ_S	$1.0 \times 10^{4}\,\mathrm{kg/m^3}$
Young's modulus	E	$5.6 \times 10^{9}\,\mathrm{N/m^2}$
Poisson ratio	ν	0.3
timestep size	Δt	$1.0 \times 10^{-5}\,\mathrm{s}$

Table 35: Multi-Cylinder testcase: physical parameters.

solvers use a relative criterion of 10^{-5}. A multi coupling scheme based on the Anderson acceleration with reused columns from ten timesteps, M-AA(10), together with a QR1 filter with $\epsilon = 10^{-6}$ is used. The simulation is carried out on the bdz partition of the CoolMAC cluster[50]. A total of six nodes à 64 cores are used – five nodes for the fluid solver and one node for all structure solvers – which results is an overall simulation time of approximately 125 hours for 0.08 s real time.

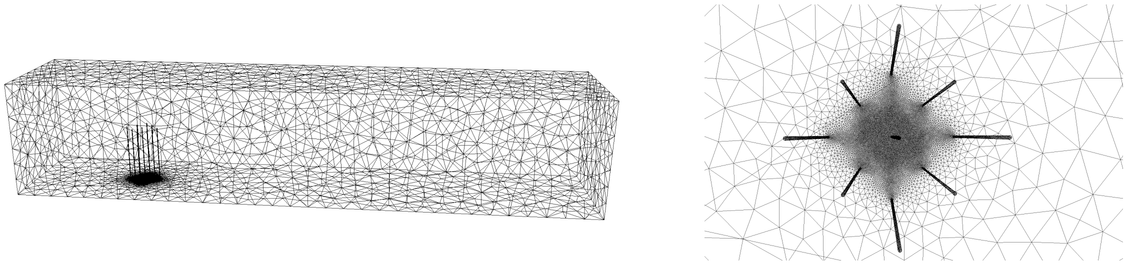


Figure 102: Fluid mesh for the Multi Cylinder testcase with nine cylinders. Wireframe (left) and close-up of the adaptive refinement (right).

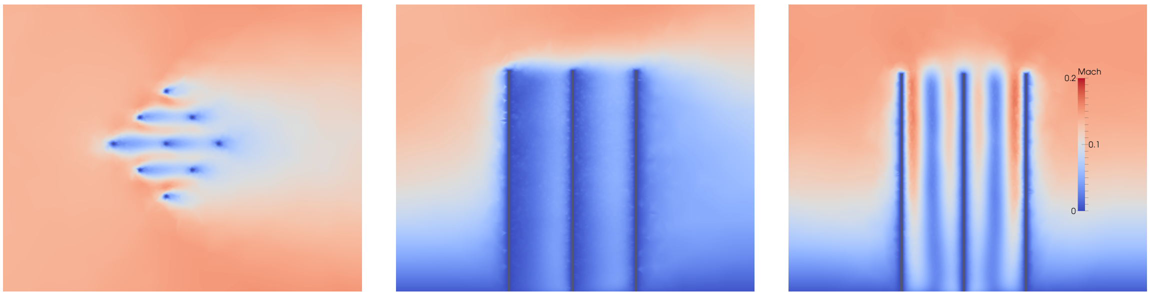


Figure 103: Multi Cylinder testcase, Mach number for different cuts through the referential (undeformed) fluid domain at $t = 0.06\,\mathrm{s}$. From left to right at the plains: $z = 0$, $y = 0$, and $x = 0.1$.

Results Most important for this thesis is probably the performance of the multi-coupling scheme. The number of sub-iterations drops from 25 to 2 over the first 13 timesteps. After timestep 168, the iterations even drop to one. The multi-coupling allows, thus, for a stable, but also for a very efficient coupling. Besides this main conclusion, I also want to briefly discuss the physical results. Figure 103 gives a first impression on the physical results. The displacements of the top center points of all cylinders over time

[50] http://www.mac.tum.de/wiki/index.php/MAC_Cluster

give more insight, as depicted in Figure 104. One can clearly see that the oscillations of the cylinders differ in magnitude and are also phase-shifted, showing the mutual influence of the cylinders. The leftmost cylinder, $C1$, shows the smallest magnitude, while the outmost cylinders of the configuration, $C4$ and $C6$, show the highest overall deformation and an oscillation around a clearly deformed state. In general, a phase-shift is visible from left to right in the cylinder configuration, meaning that the cylinders more on the right follow those on the left. For a clean study, the setup should be simulated over a longer period of time. In general, I can state, however, that the current numerical and software setup is able to simulate and compare various cylinder configurations.

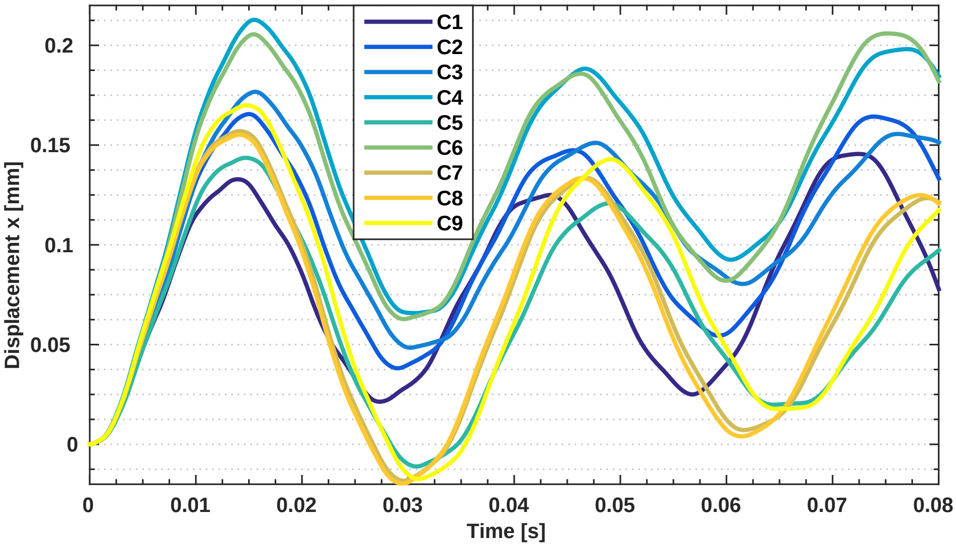


Figure 104: Multi Cylinder testcase, oscillation of the all cylinder tips. For the geometric configuration, compare Figure 101.

5.6 Uncertainty Quantification of the FSI3 Benchmark

Research in uncertainty quantification (UQ), or more concisely in the branch of forward propagation, studies how uncertainty in input parameters propagates through a physical model and influences certain key values of the output, so-called quantities of interest. Physical input parameters are often measured quantities. Their uncertainty is, therefore, a natural assumption. If we were able to determine distribution functions of quantities of interest, confidence intervals could easily be computed in a post-processing step. To make it simple, this could concern the question about the probability that the wing of an airplane does or does not break. Research in UQ has gained a lot of attention in recent years, not only to due to its necessity, but also to the growing computational possibilities.

In this showcase, I exemplarily study UQ for the FSI3 benchmark. In contrast to the classical, deterministic setting discussed in Section 3.7.1, the physical input parameters now become distributions instead of fixed deterministic values. These input parameters are the fluid and structural density, ρ_F and ρ_S, the dynamic viscosity μ, the E-module E, and the Poisson ration ν. As quantity of interest, I consider the displacement in x-direction of the backside of the cantilever – point A, please compare Figure 37 on page 55. Furthermore, stochastic moments, such as the expectation value or the variance, of the flow velocity are a natural output of an uncertain FSI simulation. I refer to the uncertain FSI3 benchmark as UFSI3.

The purpose of this showcase is not to present original UQ research, but to study how the two parallel layers developed in this thesis interplay with an additional third *sampling* layer, such as UQ. The showcase should, therefore, be regarded as a prototype, used to study the question on how far we can get with such an approach. Section 5.6.1, therefore, collects the challenges that we have to deal with in such a complex setting. The discussion comes along with a brief literature review on UQ in FSI simulations. Afterwards, Section 5.6.2 gives a compact description of the applied mathematical setup. Finally, Section 5.6.3 shows the simulation results and discusses their performance. I do not repeat a description of the FSI3 benchmark itself as it is already covered in Section 3.7.1.

5.6.1 The Multi-Challenge

In 2010, the Institute for Advanced Study of the Technical University of Munich, initiated the focus group *High Performance Computing*, which dedicated itself to *tackle the multi-challenge*[51]. The goal of the group's research is to tackle complex problems that necessitate expertise in several *multi* communities. The UFSI3 showcase falls into this category of problems. The underlying multi-physics problem and its challenges are already elaborated broadly in this thesis, recall, for example, Section 1.1. The UQ setup further introduces a multi-dimensional challenge. The five-dimensional stochastic parameter space renders brute force algorithms too costly due to the so-called *curse of dimensionality* – the exponential growth of the computational cost with the dimension. Sophisticated state-of-the-art solutions are necessary. Finally, the high computational load leads to a multi-core challenge, meaning the efficient use of modern parallel architectures. The non-intrusive stochastic collocation approach, which we apply for this showcase, allows to reuse deterministic simulations, which form an additional embarrassingly parallel layer, neglecting the pre- and post-processing. This setup is in line with the forthcoming generation of supercomputers: a vast parallel amount of simple compute nodes. To run efficiently on such an architecture, communication over nodes should be minimal, whereas the strong scalability within one node is of growing importance. Therefore, I apply the two parallel layers of this thesis – inter-solver and intra-solver parallelism – within one node, for which Section 4.5 shows good strong scalability. For the third layer – the UQ layer, the independent deterministic samples are distributed over individual nodes. Figure 105 visualizes this concept.

UQ for FSI Simulations The UFSI3 showcase is not the first UQ for FSI simulations. Most examples of the literature consider, however, simplified FSI models due to immense computational costs. Let me mention several examples: [214] studies panel flutter with the E-module of the structure modeled as a random field. The authors discretize the random field by means of a Karhunen-Loève expansion and solve the forward propagation with a pertubation method. Next, a preliminary study for uncertain cardiovascular system simulations is presented in [52]. Challenges and open research questions are

[51] http://www.tum-ias.de/focus-groups/current-focus-groups/high-performance-computing-hpc.html

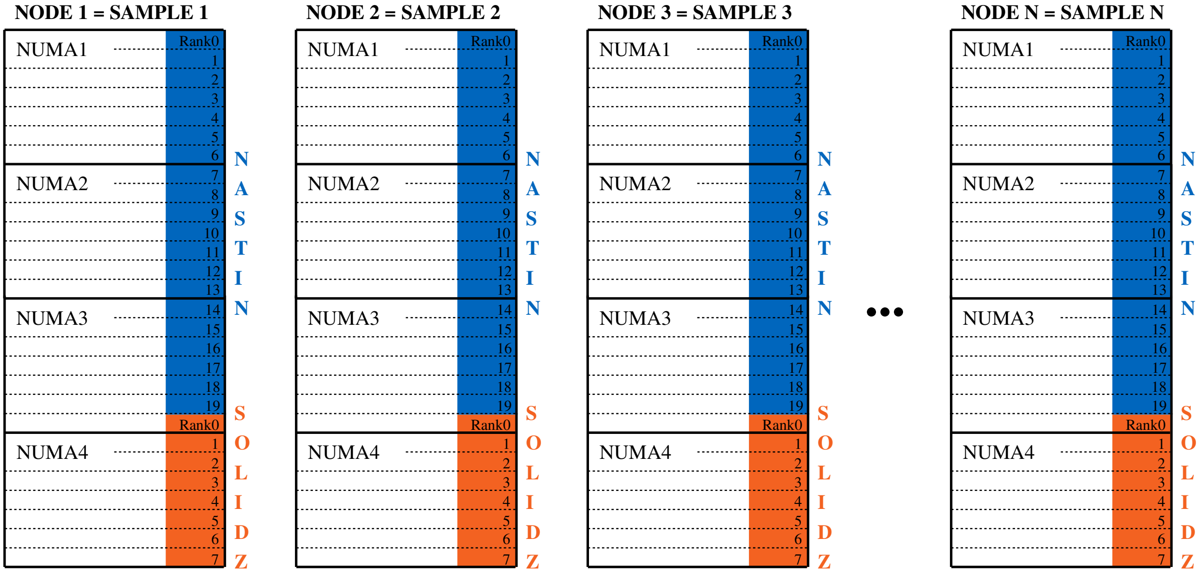


Figure 105: The three parallel layers of the UFSI3 showcase. The two parallel layers developed in this thesis – the inter-solver parallelism and the intra-solver parallelism – are applied on a single node. Furthermore, independent samples are computed in parallel on several nodes. The node layout corresponds to the Haswell architecture with 28 cores per node. This figure is also used in [80].

collected. In [233], the y-coordinate of a cylinder, immersed in flow, is modelled by a damped oscillator, whose two parameters are uncertain. The resulting UQ-FSI problem is solved by means of a generalized polynomial chaos expansion. [224] also uses a generalized polynomial chaos expansion, based on a Gram-Schmidt orthogonalization process, to analyze uncertainties in a single degree of freedom stall flutter model. The same authors study the uncertainty of a linear piston problem by employing a two-step chaos collocation approach [132]. Finally, [104] uses a Monte Carlo approach to study the stress distribution of a combustor liner. For a more general UQ overview in plain CFD simulations, I can recommend [151]. For general UQ collocation approaches, I give a brief literature review along with the mathematical setup in the next section.

5.6.2 Mathematical Setup

Since I do not intend to give an overview on the mathematical theory, I try to keep the mathematical setup as brief as possible. I assume that the reader has a basic understanding of probability theory. If not, you may refer to, e.g., [186]. The description closely follows our publication [80]. The notation is standard to allow for a smooth reading. Conflicts with other part of this thesis are not always avoidable. I first establish the stochastic setup and afterwards the sparse grid setup. Finally, both are combined to the sparse grid collocation technique.

Stochastic Collocation Stochastic collocation is based on the theory of homogeneous chaos by Wiener in 1938 [222]. In 1947, Cameron and Martin proved the convergence of the Hermite chaos expansion in terms of Gaussian random variables. This result was extended in 2002 by Xiu and Karniadakis from classical polynomial chaos to generalized polynomial chaos [232], including all polynomials of the Askey scheme. I restrict the UFSI3 showcase to Gaussian random variables, however. Thus, the associated polynomial chaos basis consists of Hermite polynomials.

Let $(\Omega, \mathcal{F}, \mathcal{P})$ be a standard probability space – Ω the sample space, $\mathcal{F}$ the σ-algebra defined on Ω, and $\mathcal{P}$ the probability measure. Furthermore, let

$$\boldsymbol{\theta} = (\theta_1, \ldots, \theta_d) : \Omega \to \mathbb{R}^d$$

denote a d-variate random vector with independent components. With bold symbols, I always refer to vectors of length d, whereas its components are in normal font and not always explicitly introduced.

For simplification, I also use $\boldsymbol{\theta}$ to directly refer to an element in $\mathbb{R}^d$, i.e., to $\boldsymbol{\theta}(\omega)$. Each component of $\boldsymbol{\theta}$ has a probability distribution function $\rho_i : \mathbb{R} \to \mathbb{R}_0^+$. Due to the independence of the components, the joint probability density function reads (see e.g. [186])

$$\boldsymbol{\rho} : \mathbb{R}^d \to \mathbb{R}_0^+, \;\; \boldsymbol{\rho}(\boldsymbol{\theta}) = \prod_{i=1}^{d} \rho_i(\theta_i) \; .$$

We consider the problem

$$f(x, \boldsymbol{\theta}), f : \mathbb{D} \times \Omega \to \mathbb{S} \; ,$$

where f is the solution of the complete FSI PDE. $\mathbb{D} \subset \mathbb{R}^4$ is the space of the deterministic variables in space and time and x the corresponding deterministic variable. $\mathbb{S}$ denotes the output space – velocity, pressure, and displacement values. $\boldsymbol{\theta}$ denotes the uncertain physical parameters. I mention further above that the dimension of the parameter space d is five. $\boldsymbol{\theta}$ being a random variable, f itself becomes one as well.

The polynomial chaos expansion approximates random variables by projecting them into a probabilistic space spanned by orthonormal polynomials $\boldsymbol{\Phi}_i$. A set of polynomials $\boldsymbol{\Phi}_i$ is called orthonormal if

$$\mathbb{E}[\boldsymbol{\Phi}_i \boldsymbol{\Phi}_j] = \int_{\Omega} \boldsymbol{\Phi}_i(\omega) \boldsymbol{\Phi}_j(\omega) d\omega = \int_{\mathbb{R}^d} \boldsymbol{\Phi}_i(\boldsymbol{\theta}) \boldsymbol{\Phi}_j(\boldsymbol{\theta}) \boldsymbol{\rho}(\boldsymbol{\theta}) d\boldsymbol{\theta} = \delta_{ij} \; ,$$

where δ_{ij} is Kronecker's delta function. Since the UFSI3 showcase is restricted to Gaussian input distributions, the associated polynomials are the *probabilists'* Hermite polynomials. The first few read

$$\Phi_0(\theta) = 1, \; \Phi_1(\theta) = \theta, \; \Phi_2(\theta) = \theta^2 - 1, \; \Phi_3(\theta) = \theta^3 - 3\theta, \; \ldots \; .$$

A simple scaling allows for normalization (see e.g. [186]). From theses univariate polynomials Φ, d-variate polynomials $\boldsymbol{\Phi}$ are constructed via the tensor product

$$\boldsymbol{\Phi}_{\boldsymbol{n}}(\boldsymbol{\theta}) = \Phi_{n_1}(\theta_1) \cdot \ldots \cdot \Phi_{n_d}(\theta_d) \; .$$

The finite-dimensional polynomial space reads

$$\mathbb{O}_P^d = \{\boldsymbol{\Phi}_{\boldsymbol{n}}(\boldsymbol{\theta}) : \sum_{i=1}^{d} n_i < P\} \; ,$$

with dimension $\binom{d+P}{d} =: N$ ([231]). For simplicity, I drop the multi-index, but use scalar index $p = 0, \ldots, N-1$ instead. The projection of $f(x, \boldsymbol{\theta})$ onto $\mathbb{O}_P^d$ gives the polynomial chaos expansion

$$f(x, \boldsymbol{\theta}) \approx f_P(x, \boldsymbol{\theta}) = \sum_{p=0}^{N-1} c_p(x) \boldsymbol{\Phi}_p(\boldsymbol{\theta}) \; ,$$

split into the stochastic contribution in the polynomials and the deterministic coefficients

$$c_p(x) = \mathbb{E}[f(x, \boldsymbol{\theta}) \boldsymbol{\Phi}_p(\boldsymbol{\theta})] = \int_{\mathbb{R}^d} f(x, \boldsymbol{\theta}) \boldsymbol{\Phi}_p(\boldsymbol{\theta}) \boldsymbol{\rho}(\boldsymbol{\theta}) d\boldsymbol{\theta}, \;\; p = 0, \ldots, N-1 \; . \tag{8}$$

The moments of $f_P(x, \boldsymbol{\theta})$ can directly be computed from the coefficients (e.g. [186]),

$$\mathbb{E}[f_P(x, \boldsymbol{\theta})] = c_0(x) \;\text{ and }\; \mathrm{Var}[f_P(x, \boldsymbol{\theta})] = \sum_{p=1}^{N-1} c_p^2(x) \; .$$

Since f is unknown, the coefficients' integrals cannot be computed directly. An approximation via a quadrature rule, however, leads to a series of simple deterministic evaluations of $f(x, \boldsymbol{\theta}(\omega))$, for fixed $\omega \in \Omega$. The remaining problem is the construction of such a high-dimensional quadrature rule. A simple tensor construction building on one-dimensional quadrature rules lets the amount of deterministic evaluations grow exponentially with d, yielding an unfeasible computational cost. The sparse grid construction of the next paragraph, however, mitigates this *curse of dimensionality* and makes the computational cost feasible.

Sparse Grid Interpolation For the UFSI3 showcase, we do not use a direct sparse grid quadrature, but substitute f in (8) by a sparse grid interpolant, which can then be integrated analytically. [80] also discusses approaches based on a direct sparse grid quadrature and compares them to the interpolation approach. Furthermore, another alternative is to not use the polynomial chaos expansion at all, but directly construct f as a sparse grid interpolant [96]. In the following, we construct the sparse grid interpolant on the unit cube $[0;1]^d$. To transform f from $\mathbb{R}^d$ to the unit cube, we use the component-wise cumulative distribution function

$$\boldsymbol{F} : \mathbb{R}^d \to [0;1]^d \text{ with } F_i(\tilde{\theta}_i) = \int_0^{\tilde{\theta}_i} \rho_i(\theta_i) d\theta_i \ .$$

The transformed problem then reads

$$\hat{f}(x, \boldsymbol{u}) := f(x, \cdot) \circ \boldsymbol{F^{-1}}(\boldsymbol{u}) \ ,$$

where $\boldsymbol{u} \in [0;1]^d$ denotes the transformed parameters in the unit cube.

For the moment, let us assume that $\hat{f}$ vanishes on the boundary of the unit cube. The hierarchical construction of the sparse grid spaces starts with the one-dimensional hat functions

$$\varphi_{l,k}(u) = \phi\left(\frac{u - kh_l}{h_l}\right) \text{ with } h_l = 2^{-l} \text{ and } \phi(u) = \max(1 - |u|, 0) \ , \quad u \in [0;1] \ .$$

Yet again, the d-variate version is defined as the tensor product

$$\varphi_{\boldsymbol{l},\boldsymbol{k}}(\boldsymbol{u}) = \prod_{i=1}^{d} \varphi_{l_i,k_i}(u_i) \ .$$

Furthermore, we use the index set

$$\mathcal{I}_{\boldsymbol{l}} = \{(\boldsymbol{l}, \boldsymbol{k}) \in \mathbb{N}^d \times \mathbb{N}^d, k_i \text{ odd } \wedge 1 \leq k_i \leq 2^{l_i} - 1, \ \forall i = 1 \dots d\}$$

to define the hierarchical increment spaces

$$W_{\boldsymbol{l}} := \operatorname{span}(\{\varphi_{\boldsymbol{l},\boldsymbol{k}}, (\boldsymbol{l}, \boldsymbol{k}) \in I_{\boldsymbol{l}}\}) \ .$$

The regular sparse grid space of level $L \in \mathbb{N}$ now only considers those subspaces that contribute most to the overall solution [40],

$$V_L = \bigotimes_{|\boldsymbol{l}|_1 \leq L+d-1} W_{\boldsymbol{l}} \ .$$

The number of points in V_L is $\mathcal{O}\left(2^L (\log(2^L))^{d-1}\right)$, which is much smaller than the size of the associated full grid $\bigotimes_{\boldsymbol{l}, l_i \leq L} W_{\boldsymbol{l}}$, $\mathcal{O}(2^{Ld})$ [40]. The exponential growth with the dimension, in particular, is shifted to a logarithmic term. To better understand this concept, Figure 106 visualizes the regular sparse grid for level $L = 5$ and dimensions two and three.

The sparse grid interpolant of $\hat{f}(x, \boldsymbol{u})$ is denoted by $f_{\mathcal{I}}(x, \boldsymbol{u}) \in V_L$ and reads

$$f_{\mathcal{I}}(x, \boldsymbol{u}) = \sum_{\substack{|\boldsymbol{l}|_1 \leq L+d-1 \\ (\boldsymbol{l},\boldsymbol{k}) \in \mathcal{I}_{\boldsymbol{l}}}} \alpha_{\boldsymbol{l},\boldsymbol{k}}(x) \varphi_{\boldsymbol{l},\boldsymbol{k}}(\boldsymbol{u}) \ . \tag{9}$$

$\alpha_{\boldsymbol{l},\boldsymbol{k}}(t)$ are the so-called hierarchical surpluses and can be explicitly computed from the interpolation conditions via point-wise evaluations of $\hat{f}(x, \cdot)$. Further above, I mention that the sparse grid space only includes those hierarchical increment spaces that contribute most. Mathematically, this means that the sparse grid subset is optimal in the L^2 sense [40, 237],

$$\|\hat{f}(x, \boldsymbol{u}) - f_{\mathcal{I}}(x, \boldsymbol{u})\|_{L^2} \in \mathcal{O}\left(h_L^2 (\log(h_L^{-1}))^{d-1}\right) \ .$$

An only logarithmic degeneration from the full grid convergence rate $\mathcal{O}\left(h_L^2\right)$ is achieved.

Since $\hat{f}(x, \cdot)$ does not actually vanish on the boundary, additional measures need to be taken. Including additional degrees of freedom on the boundary becomes increasingly expensive with increasing dimensionality and is, therefore, no solution. Instead, we use modified basis functions, which correspond to a

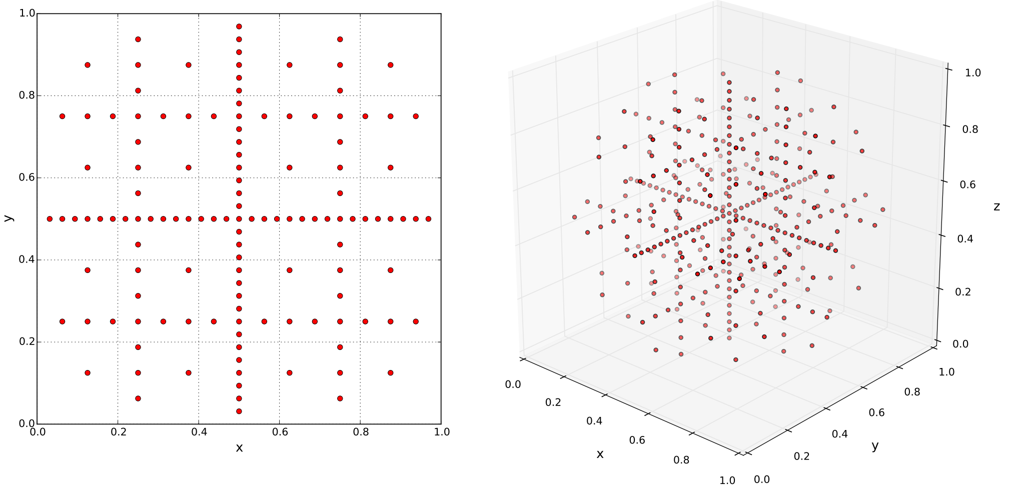


Figure 106: Regular sparse grid construction for level $L = 5$ and dimensions two (left) and three (right). Each marked point corresponds to the tip of a hat function. This figure is also used in [80]

linear extrapolation at the boundary and are discussed in [161]. For the sake of brevity, details of the construction are not shown. The reader may refer to [80].

The hierarchical structure of the sparse grid space allows for a natural adaptivity in the stochastic space. The hierarchical surplus can be used as a metric for the local interpolation error. The surplus, however, still depends on x, the deterministic variable. To reduce the surplus to a simple real value, an integration over the complete deterministic domain would be one solution, but can be cumbersome. A better and simpler strategy is to directly use a quantity of interest. For the UFSI3 showcase, this is a simple point value, the x-displacement of the backside of the cantilever at a certain point in time, as mentioned further above. All hierarchical errors estimators are now sorted by their absolute value. Afterwards, hierarchical descendants of the highest 10% are added. If not all hierarchical parents already exist in the refined grid, they are added as well. For details, please refer to [161]. Finally, $\hat{f}(x, \cdot)$ is evaluated at the newly created points. The complete procedure is repeated several times.

Combining Both Worlds To combine the sparse grid and the collocation concepts, the interpolant (9) is inserted into the coefficients' formula (8). The transformation factors from the substitution and from the change of probability measure cancel each other out, please compare [231] or Lemma 1 in [96]. We get[52]

$$\begin{aligned}
c_p(x) &= \int_{[0,1]^d} f(t, \boldsymbol{F}^{-1}(\boldsymbol{u}))\boldsymbol{\Phi}_p(\boldsymbol{F}^{-1}(\boldsymbol{u}))d\boldsymbol{u} \\
&\approx \int_{[0,1]^d} \Big(\sum_{\substack{|\boldsymbol{l}|_1 \le n+d-1 \\ (\boldsymbol{l},\boldsymbol{k}) \in \mathcal{I}_{\boldsymbol{l}}}} \alpha_{\boldsymbol{l},\boldsymbol{k}}(x)\varphi_{\boldsymbol{l},\boldsymbol{k}}(\boldsymbol{u}) \Big) \boldsymbol{\Phi}_p(\boldsymbol{F}^{-1}(\boldsymbol{u}))d\boldsymbol{u} \\
&= \sum_{\substack{|\boldsymbol{l}|_1 \le n+d-1 \\ (\boldsymbol{l},\boldsymbol{k}) \in \mathcal{I}_{\boldsymbol{l}}}} \alpha_{\boldsymbol{l},\boldsymbol{k}}(t) \prod_{i=1}^{d} \int_{[0,1]} \Phi_i(F_i^{-1}(u_i))\varphi_{l_i,k_i}(u_i)du_i \;.
\end{aligned}$$

The resulting one-dimensional integrals can be computed analytically and a-priori. As already mentioned above, everything else can be computed from theses coefficients: statistical moments, the distribution of the quantity of interest, confidence intervals, and so forth.

[52] For the sake of simplicity, the formula does not consider the stochastic adaptivity.

5.6.3 Results

All sample runs were performed on the Haswell partition of SuperMUC, second phase. Each node consists of 28 cores. Please recall the hardware description I give at the beginning of Chapter 4. The pre- and postprocessing is run on a simple workstation and does not contribute significantly to the the overall computational load (see also [79]). The numerical setting is identical to the deterministic simulation in Section 3.7.1 using the fine mesh configuration. We use J-AA-20 as coupling scheme, meaning a parallel Anderson acceleration with reused columns from 20 previous timesteps. The single-physics solvers are Alya Nastin and Solidz. All five input parameters follow a normal distribution around their deterministic values with a relative standard deviation of 15% of the mean. The mean values are: fluid density $\rho_F = 1.0 \times 10^3\,\mathrm{kg/m^3}$, structural density $\rho_S = 1.0 \times 10^3\,\mathrm{kg/m^3}$, dynamic viscosity $\mu = 1.0\,\mathrm{kg/(m\,s)}$, Young's modulus $E = 5.6 \times 10^6\,\mathrm{N/m^2}$, and the Poisson ratio $\nu = 0.4$.

Computational Load I first compare the runtime of several deterministic runs to find the optimal load balancing between fluid and structure solver within one node. I, therefore, consider the runtime after 50 timesteps. Table 36 lists the respective numbers. A distribution of 20 fluid cores and 8 structure cores appears to be optimal. The stochastic simulation uses 3.000 timesteps with a timestep size of $1.0 \times 10^{-3}\,\mathrm{s}$. The polynomial chaos expansion is cut after five terms, $P = 4$, which corresponds to $N = 126$. For the sparse grid computations, the software SG++[53] is used. For the adaptivity, the x-displacement of the cantilever is measured at the last timestep. The adaptive procedure starts with level $L = 2$, comprising 11 samples and refines three times for additional 10, 36, and 84 samples, summing up to a total of 141 samples. Figure 107 shows a histogram of the runtimes of all independent samples. Since all samples feature slightly different added-mass effects, they are also expected to differ in runtime. All samples lie, however, between 10.1 h and 12.3 h, which indicates a satisfactory load balancing over all nodes. As an outlook to UQ simulations of more complex FSI scenarios, I want to stress two key numbers. First, the total compute load of the UFSI3 showcase, which is approximately $11 \cdot 28 \cdot 141 \approx 43000$ compute hours if the average sample takes 11 h to compute. Thus, a good strong scalability of a single FSI simulation within one node is mandatory. Here, the two parallel layers of this thesis can be applied successfully. Also, the applied UQ approach should carefully decide upon each additional sampling point due to the computational load of each single point. Second, the four adaptivity cycles of simulations, which need to be computed after each other, lead already to an approximate overall runtime of 44 h, neglecting the non-perfect load balancing. A UQ approach without an additional parallel layer would, thus, lead to a tremendous overall runtime. To summarize, only the combination and efficient usage of all three parallel layers lead to an acceptable overall runtime for UQ of FSI simulations.

Cores F	Cores S	Time[s]
24	4	1163.95
23	5	1073.49
22	6	1087.84
21	7	1010.30
20	**8**	**1007.44**
19	9	1026.28

Table 36: Load balancing within one node: runtimes of a determinstic simulation of 50 timesteps for various core distributions. The fluid solver Nastin uses 69.460 elements and the structure solver 15.850 elements.

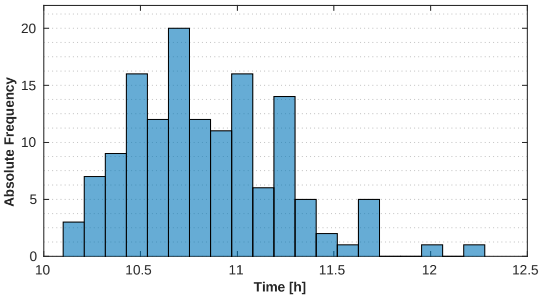


Figure 107: Load distribution among all nodes: histogram of the runtime (3.000 timesteps) of all independent samples. For comparison: the runtime of the deterministic FSI3 scenario is 10.81 h.

Stochastic Results Although not being the main focus of this showcase, I also want to briefly mention the stochastic results of the UFSI3 showcase. Figure 108, top, compares the time evolution of the x-displacement of the cantilever tip of the deterministic simulation with the expectation value of the UQ

[53] `http://sgpp.sparsegrids.org`

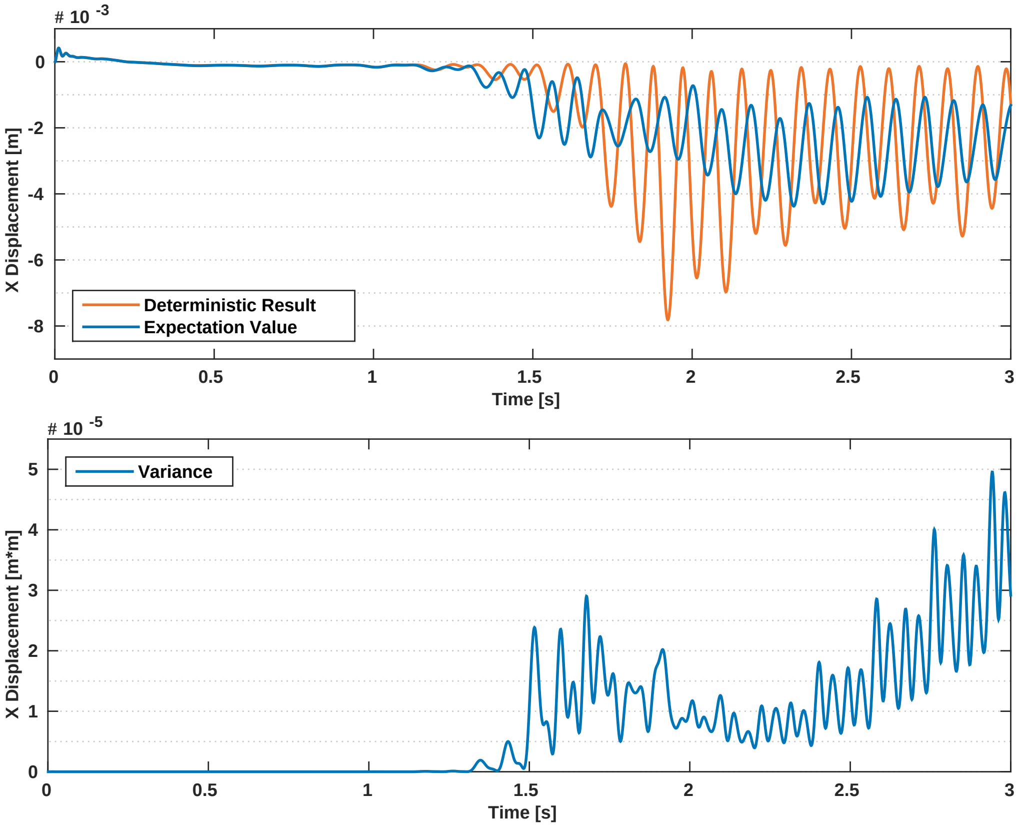


Figure 108: Time evolution of the x-displacement of the cantilever's tip. Stochastic results are compared to the associated deterministic simulation. Top: Expectation value, bottom: variance. This figure is also used in [80].

	Mean	Amplitude	Frequency
Deterministic Result	0.0023	-0.0025	10.823
Expectation	0.0024	-0.0026	10.829
Variance	1.5808e-07	1.8166e-07	0.0655

Table 37: Mean, amplitude and frequency of the UFSI3 oscillation. All values are computed after the initial phase, from 2.4 s on.

simulation. The expectation values show a significantly smaller amplitude. This is assumably due to the super-position of various oscillations, which inhibit each other. An important conclusion from this observation is that UQ results of oscillations should not be regarded in this time evolution fashion. More meaningful is to directly look at the mean, amplitude and frequency of the oscillation, as listed in Table 37. The expectation values are close to the deterministic values. Please note that they do not have to match, in general, as the overall FSI problem is a non-linear problem. An interesting observation is the higher variance of the frequency, also if normalized by the expectation value, compared to the mean and amplitude of the oscillation. Figure 108, bottom, also shows the time evolution of the variance. Again, several oscillation are super-posed. Still, the variance increases over time, as expected. Finally, Figure 109 shows the expectation value and variance of the velocity magnitude in the complete fluid domain at the last timestep. The super-posed oscillations are clearly visible. Furthermore, as expected, the highest variance is visible in the wake of the cantilever. To properly analyze the influence of each input parameter, a variance based global sensitivity analysis can be conducted. For the sake of brevity, I do not show the analysis here, but refer the reader to our work in [80].

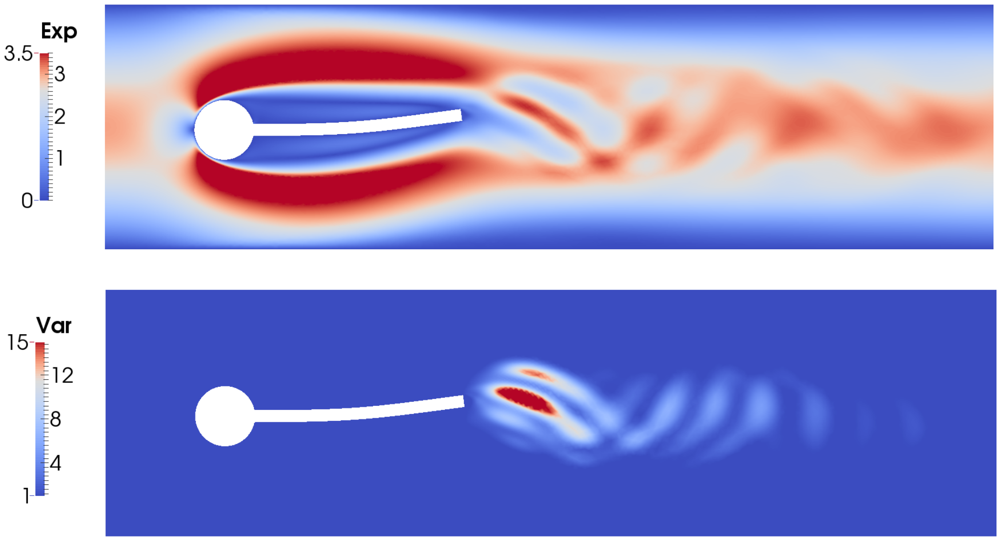


Figure 109: Expectation value (top) and variance (bottom) of the velocity magnitude at the last timestep, $t = 3.0\,\mathrm{s}$ of the UFSI3 showcase. This figure is also used in [80].

Summary of Chapter 5

- The applicability and importance of the two newly-developed parallel layers is exemplarily shown by means of several showcases.
- The Aorta showcase simulates blood flow through an aortic geometry. Parallel coupling schemes allow for an overall speed-up compared to sequential ones while requiring less resources.
- The Jet scenario and the Bending Tower scenario both show a technically successful three-field coupling – the Jet scenario a flow coupling between different models and the Bending Tower an actual fluid-structure-acoustics interaction. With the combination of explicit and implicit coupling schemes, it is possible to arrive at an overall inter-solver parallelism, also with three solvers. A static load-balancing is exemplarily studied for the Jet scenario.
- The Hemisphere showcase shows preliminary results for a highly-asymmetric turbulent FSI. The fully peer-to-peer concept of preCICE allows for an efficient coupling. The Anderson acceleration is able to couple turbulent FSI successfully.
- The Multi-Cylinder showcase simulates the flow around multiple thin cylinders as a simplified model of brush seals. The fully-implicit multi-coupling concept leads to a very efficient coupling between mutliple solvers.
- The UFSI3 showcase shows the successful combination of the two newly-developed layers with a third uncertainty quantification layer.
- The overall range of various physics and applied solvers clearly shows the flexibility and the applicability of the partitioned approach and the software and parallelization concepts of preCICE.

6 Conclusions

To conclude this thesis, I recapitulate its main contributions and discuss future challenges. First, I revisit the main findings of this thesis in Section 6.1. Here, I focus on the two newly introduced parallel layers and their impact. Afterwards, Section 6.2 gives an outlook on two particular future topics: interpolation in time and exascale computing.

6.1 Summary of the Thesis

To advance research in the most challenging multi-physics problems, more and more physical effects are included into simulations. At the same time, the reuse of sophisticated legacy codes is a natural desire. To allow for both, an inherently flexible and scalable simulation environment is indispensable.

This thesis studies fluid-structure interaction (FSI) as a prototype of a challenging multi-physics problem, but tries to deduce general techniques. As the starting point of my thesis, I consider the coupling library preCICE, which already offers great flexibility by means of its partitioned black-box approach and its high-level application interface. The library offers methods for interpolation between non-matching coupling interfaces, fixed-point acceleration schemes and means for communication between independent executables. Chapter 2 gives an introduction to preCICE. I overcome the scalability limitations of the original version [99] by introducing parallelism on two levels: inter-solver parallelism requiring novel numerical schemes in Chapter 3 and intra-solver parallelism requiring new computer science approaches in Chapter 4. Please recall Figure 3 on page 6 for a visualization of both layers. Both new parallel layers do not require any changes to the application programming interface of preCICE and do, thus, not degenerate the existing flexibility.

Inter-Solver Parallelism Classical implicit black-box coupling algorithms are based on a staggered execution of the fluid and the structure solver, one after the other. If both solvers use their own computational resources, this can – on average over the total runtime – lead to idling of up to half of the processors. Sharing resources among both solvers is, in general, no solution, as the high asymmetry of FSI problems leads to different scalability constraints of both solvers. The only clean solution are parallel coupling schemes, which allow for the simultaneous execution of both solvers. Such a parallelization of a coupling scheme should, however, not degenerate its convergence speed. One of the main contributions of this thesis is the development of parallel coupling schemes in Chapter 3. Their construction is based on the observation that classical staggered coupling schemes, such as the IQN-ILS scheme [60], can be decomposed into a staggered block-Gauss-Seidel fixed-point equation and a fixed-point equation solver, such as the Anderson acceleration [1]. Substituting the staggered fixed-point equation by a parallel block-Jacobi one results in a parallel coupling scheme. To this end, a weighting of the outputs of both solvers is necessary. In Section 3.6, I show that a scaling with the sum of the squares of previous values is an efficient, yet robust choice. Another contribution of Chapter 3 is the usage of a generalized Broyden scheme [78] instead of the Anderson acceleration. This gives a scheme that renders the tuning of reused past values unnecessary, but requires an explicit storage of the Jacobian instead.

By means of several benchmarks, I show in Section 3.7 that parallel coupling schemes result in the same convergence speed as their associated sequential counterparts. In Section 5.2, I use the Aorta showcase to exemplarily show that parallel coupling schemes lead indeed to an overall speed-up while needing less resources.

Finally, in Chapter 3, I also study the generalization of parallel coupling schemes to fully-implicit coupling of more than two solvers. A simple combination of classical coupling schemes between two solvers does not allow for such a robust coupling. I show this by means of several simple scenarios in Section 3.8. The Multi-Cylinder showcase in Section 5.5, finally, shows the applicability of the multi-coupling scheme for a complex application.

Intra-Solver Parallelism Many coupling software approaches use a centralized entity, often called a server, which stores the coupling data, computes the interpolation between coupling meshes, and applies fixed-point acceleration schemes. Such an overall software layout becomes a bottleneck for larger scenarios due to several reasons: reading and writing data requires a 1:N communication and the data mapping and fixed-point acceleration are executed as a serial computation. Furthermore, all coupling meshes need to be communicated, which is, in particular, annoying for highly asymmetric cases. The original version of preCICE already features a peer-to-peer layout [99], but based on a server for each solver. In this thesis, I port preCICE to a fully parallel peer-to-peer layout, without any central entity. To this end, I develop a mesh re-partitioning strategy (Section 4.1) and execute all three feature groups of preCICE – communication, interpolation, and fixed-point acceleration – on distributed data (Sections 4.2 to 4.4). I follow the general guideline to make the coupling effort per timestep as small as possible, while keeping the initialization time tolerable. This guideline is justifiable for a large amount of timesteps, compared to a single initialization. The showcases of Chapter 5 show that this holds true for a variety of applications. In Chapter 4, I study the performance of each feature group of preCICE separately and afterwards the applicability of the concepts on two physical scenarios: a travelling pulse in an Euler domain for up to 16,000 cores and the highly asymmetric FSI benchmark PfS-1a for up to 6,272 cores. The coupling effort always remains negligible and does, therefore, not interfere with the overall scalability. The intra-solver parallelism is of importance for all showcases of Chapter 5.

Together, all five showcases of Chapter 5 make two things clear. First, the flexibility of the partitioned approach and its realization in preCICE can be applied successfully. The showcases cover various physical setups and different solvers. Even more than two solvers are applied successfully in Sections 5.3 and 5.5. Second, both parallel layers developed in this thesis can be applied successfully. Thus, the coupling of single-physics codes is possible without degenerating their scalability. With the developments of this thesis, massively parallel simulations of various multi-physics problems are possible.

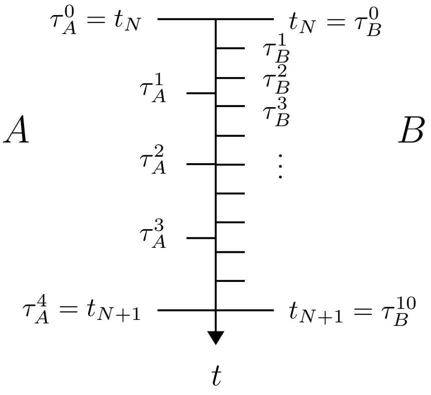
6.2 Future Challenges of Partitioned Multi-Physics

Although, with the achievements of this thesis, the applicability of preCICE can be increased from moderately to massively parallel setups and, therefore, to a whole new range of applications, certain scenarios require extensions. In this section, I discuss two of them: interpolation in time to better resolve multi-scale phenomena and technical challenges on the way to exascale computing.

Interpolation in Time Both scenarios of Section 5.3 reveal the limitation of the current time treatment in preCICE. The Jet scenario in Section 5.3.2 encompasses three flow domains whose timestep restrictions differ up to one order of magnitude. preCICE is technically able to handle subcycling, meaning that one solver A performs smaller local timesteps till another solver B finishes one bigger global timestep. In this case, Solver A uses, however, the value from solver B as constant boundary condition throughout all local timesteps, thus a order-zero extrapolation for parallel coupling schemes. For the Jet scenario, this leads to instabilities. We, thus, have to use identical timestep sizes in all three flow domains, which artificially increases the computational cost. The 3D Bending Tower scenario in Section 5.3.3 shows a similar problem. The compressible flow solver of OpenFOAM uses a second order, implicit time integration scheme. The acoustic solver of Ateles is second or forth order and explicit in time. With the current coupling approach, the overall time integration order, however, degenerates to one. Two would be desirable. Furthermore, the explicit-implicit combination would again allow us to use different timestep sizes.

The solution to both problems could be based on boundary representations of both solver that are continuous in time during one global timestep $t_N \rightsquigarrow t_{N+1}$. Let $f_A, f_B : [t_N, t_{N+1}] \to \mathbb{R}^n$ be such continuous boundary representations (CBRs). Spectral deferred correction [76] allows, for example, to construct such a CBR in an iterative manner, meaning that a zero-order initial solution is improved by one order in each iteration. [30] already discusses the closely related integral deferred correction for FSI. The combination of f_A and f_B with implicit coupling could solve both issues from above. To illustrate this, let us the consider the following example. Let us assume that solver A uses a forth-order Runge-Kutta scheme with local timesteps $\tau_A^0 = t_N, \tau_A^1, \ldots, \tau_A^4 = t_{N+1}$ and B subcycles with a constant ten-times smaller local timestep size, $\tau_B^0 = t_N, \tau_B^1, \ldots, \tau_B^{10} = t_{N+1}$. Both solvers hold a CBR, which is iteratively improved, $f_A^k \rightsquigarrow f_A^{k+1}$ and $f_B^k \rightsquigarrow f_B^{k+1}$, respectively. During coupling iteration k, solver A now uses f_B^k to evaluate boundary conditions at its necessary local timesteps τ^A, and the other way around. Algorithm 4 depicts this procedure. If we assume that f indeed improves by order in every iteration, both solvers can reach their necessary order and the subcycling is based on a meaningful extrapolation.

Algorithm 4 Coupled time integration of arbitrary order via continuous boundary representations (CBRs) and implicit coupling. $f_A, f_B : [t_N, t_{N+1}] \to \mathbb{R}^n$ denote the CBRs, which are iteratively improved $k \rightsquigarrow k+1$. The substeps of both solvers can be computed in parallel to each other. A and B exemplarily use four and ten time local timesteps τ_A and τ_B, respectively.



```
start with constant approximations,
        $f_A^0 \equiv f_A(t_N)$ and $f_B^0 \equiv f_B(t_N)$
for $k = 1 \dots$ do
    compute substeps of $A$ with BC
        $f_B^k(\tau_1^A), f_B^k(\tau_2^A), \dots, f_B^k(\tau_4^A)$
    compute substeps of $B$ with BC
        $f_A^k(\tau_1^B), f_A^k(\tau_2^B), \dots, f_A^k(\tau_{10}^B)$
    improve $f_A^k \rightsquigarrow f_A^{k+1}$ and $f_B^k \rightsquigarrow f_B^{k+1}$
    fixed-point acceleration
end for
```

The technical problem how to publish the CBR to the opposite solver remains. Two general solutions are possible. First, each solver computes its CBR by itself. Then, extensions to the API of preCICE are necessary to publish the local timesteps τ_A and τ_B and to retrieve the associated interface values. The second solution would require preCICE to compute the CBRs. Then, no changes to the API would be necessary, but boundary conditions could simply be retrieved the usual way. The method to compute the CBR, however, would have to be a black-box technique, meaning that it needs to work independently from the solvers' time discretization techniques. Which one of the two solutions is preferable probably depends on how well such a black-box CBR could be computed. This is an open research question.

Roadmap to Exa-Scale For the FSAI showcase in Section 5.3, I already mention the ExaFSA project. It is part of the German priority program on exascale computing SPPEXA, which was initiated to prepare, among other tasks, simulation software to cope with the challenges of the upcoming exascale era (as, e.g., summarized in [71]). Many concepts of this thesis can scale up to approximately 10,000 cores, but not further. In particular, this concerns the initialization phase. If a large amount of timesteps is required compared to a single initialization phase, as it is the case for all showcases in Chapter 5, the initialization effort is negligible. The picture changes, however, if multiple re-initialization phases become necessary to cope with changing coupling interfaces. An purely Eulerian fluid solver (with moving boundaries), solvers with dynamic adaptivity or a dynamic load-balancing between all solvers, all entail such changing interfaces.

The main building block for a faster initialization would be a hierarchical mesh re-partitioning concept. The current implementation uses a gather-scatter concept, which entails not only a runtime bottleneck, but also a strict memory restriction. A hierarchical concept should first match only bounding boxes of the interface partitions of both solvers, such as applied in, for example, [73, 163, 184]. A preliminary M2N communication would be necessary. Only in a second step, actual vertices are compared, now in a completely local fashion. Afterwards the M2N could be reduced to the remaining overlaps.

Furthermore, there are several smaller work packages to speed up the initialization. The point-to-point communication could be constructed directly from local information without an additional broadcast of the vertex distributions. The nearest-neighbor search could be reduced to linear complexity by sorting all vertices into a coarse mesh structure. Publishing connection information for the kernel 1:N communication channels currently uses the file system – a fact that becomes problematic beyond approximately 1,000 connections. Directly using MPI routines such as `MPI_Publish_name` and `MPI_Lookup_name` should be a remedy. The MPI routine for creating the communicators, on the other hand, currently shows performance and robustness issues. For the intra-solver communicator, reusing the solver's own communicator would be a straight-forward alternative, but has to be carefully implemented not to interfere with the flexibility of preCICE – an MPI-free usage of preCICE should still be possible. For the creation of inter-solver communicators, the linear increase in runtime (in the number of cores per participant) has to be studied thoroughly, in particular since the TCP/IP communication does not show a similar overhead.

Besides the initialization, also two specific building blocks per timestep show scalability limitations. First, the radial basis function mapping suffers from the global polynomial rows, which require global communication. Mapping without these rows is, however, possible and has to be carefully studied. Alternatives that also discuss fully local RBF schemes are, for example, [67, 195]. Second, the generalized Broyden scheme features quadratic complexity, both in computation and memory. A low-rank approximation can, however, solve both [177]. All other building that are executed per timesteps are ready for exascale, including projection-based mappings, the M2N communication, and the Anderson acceleration.

Let me summarize this last chapter again. This thesis significantly increases parallelism for partitioned multi-physics simulation on two levels without degenerating the inherent flexibility of the partitioned approach. The flexibility allows for easy extensions of further building blocks – further physical effects, further numerical approaches, and further HPC optimizations. Thus, an important milestones towards the most complex multi-physics applications is achieved.

Acknowledgements

I want to take the opportunity to briefly thank those who had an important impact on this thesis. I was in the lucky position to have two thesis supervisors and I highly benefited from both. Thank you, Miriam, for all the honest and direct advice throughout these four years, for always having time and always having the right words for me. Thank you, Hans, for the unconditional support for whatever I suggested and for the open-minded work environment at our chair. I mentioned on several occasions throughout this thesis that preCICE is a joint software project. I want to thank my colleagues for the fruitful collaboration and for always pursuing a shared aim. Thank you, Bernhard, Florian and Klaudius. The ExaFSA project had an important influence on this work. Thank you, Verena and David, and everybody else. Also, many students contributed to this work, among even some who later joint our group as colleagues. Thank you, Alexander S., Alexander R., Ionut and Klaudius again.

My thesis further highly benefited from two research stays abroad. First, I had the chance to continue the collaboration with the CASE department of the Barcelona Supercomputing Center, which already started with my Diploma Thesis. Thank you for the warm welcome again and the support at all time, Daniel, Guillaume, Herbert, Juan-Carlos, Mariano, Miguel, and everybody else. Second, I spent two months at the Lawrence Livermore National Laboratory. Thank you, Carol, for making this possible and for all the highly interesting discussions. Thank you, John and Matthieu, and everybody else in Livermore. Other collaborations also contributed to this thesis – thank you, Alexander F. from LFA, Aditya and collegues from STATIK, and Robby Haelterman. Furthermore, thank you for proof-reading, Alex, Berni, Bri, JCC, and Verena.

Finally, I thankfully acknowledge the financial support of the priority program 1648 – Software for Exascale Computing of the German Research Foundation and of the Institute for Advanced Study of the Technical University of Munich as well as the provided computing time on the SuperMUC and the CoolMAC at the Leibniz Supercomputing Centre.

References

[1] D. Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM*, 12(4):547–560, 1965. (Cited on pages 8, 19, 29, 34, 37, and 135.)

[2] D. Anderson. Comments on Anderson acceleration, mixing and extrapolation. ICERM Workshop: Numerical Methods for Large-Scale Nonlinear Problems and Their Applications, 2015. (Cited on page 35.)

[3] A. Atanasov. *Software Idioms for Component-based and Topology-aware Simulation Assembly and Data Exchange in High Performance Computing and Visualisation Environments.* PhD thesis, Institut für Informatik, Technische Universität München, 2014. (Cited on page 24.)

[4] S. Badia, F. Nobile, and C. Vergara. Fluid-structure partitioned procedures based on Robin transmission conditions. *Journal of Computational Physics*, 227(14):7027–7051, 2008. (Cited on page 5.)

[5] S. Badia, A. Quaini, and A. Quarteroni. Splitting methods based on algebraic factorization for fluid-structure interaction. *SIAM Journal on Scientific Computing*, 30(4):1778–1805, 2008. (Cited on page 5.)

[6] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, S. Zampini, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 – Revision 3.6, Argonne National Laboratory, 2015. (Cited on pages 18 and 88.)

[7] D. Balzani, S. Deparis, S. Fausten, D. Forti, A. Heinlein, A. Klawonn, A. Quarteroni, O. Rheinbach, and J. Schröder. Numerical modeling of fluid–structure interaction in arteries with anisotropic polyconvex hyperelastic and anisotropic viscoelastic material models at finite strains. *International Journal for Numerical Methods in Biomedical Engineering*, 2015. (Cited on pages 2 and 5.)

[8] J. W. Banks, W. D. Henshaw, and D. W. Schwendeman. An analysis of a new stable partitioned algorithm for FSI problems. part i: Incompressible flow and elastic solids. *Journal of Computational Physics*, 269:108–137, 2014. (Cited on page 5.)

[9] A. T. Barker and X.-C. Cai. Scalable parallel methods for monolithic coupling in fluid-structure interaction with application to blood flow modeling. *Journal of Computational Physics*, 229(3):642–659, 2010. (Cited on pages 3 and 5.)

[10] P. Bastian, G. Buse, and O. Sander. Infrastructure for the coupling of dune grids, 2010. (Cited on page 22.)

[11] K. Bathe, C. Nitikitpaiboon, and X. Wang. A mixed displacement-based finite element formulation for acoustic fluid-structure interaction. *Computers & Structures*, 56(2):225–237, 1995. (Cited on page 1.)

[12] Y. Bazilevs, M.-C. Hsu, J. Kiendl, R. Wüchner, and K.-U. Bletzinger. 3D simulation of wind turbine rotors at full scale. Part II: Fluid-structure interaction modeling with composite blades. *International Journal for Numerical Methods in Fluids*, (65):236–253, 2010. (Cited on pages 1 and 5.)

[13] Y. Bazilevs, M.-C. Hsu, and M. Scott. Isogeometric fluid–structure interaction analysis with emphasis on non-matching discretizations, and with application to wind turbines. *Computer Methods in Applied Mechanics and Engineering*, 249:28–41, 2012. (Cited on page 18.)

[14] Y. Bazilevs, K. Takizawa, and T. E. Tezduyar. *Computational Fluid-Structure Interaction: Methods and Applications.* John Wiley & Sons, 2012. (Cited on pages 5 and 11.)

[15] A. Beckert and H. Wendland. Multivariate interpolation for fluid-structure-interaction problems using radial basis functions. *Aerospace Science and Technology*, 5(2):125–134, 2001. (Cited on page 18.)

[16] J. Benk. *Immersed Boundary Methods within a PDE Toolbox on Distributed Memory Systems.* PhD thesis, Technische Universität München, 2012. (Cited on page 6.)

[17] J. Berland, C. Bogey, and C. Bailly. Numerical study of screech generation in a planar supersonic jet. *Physics of Fluids*, 19(7), 2007. (Cited on page 115.)

[18] C. Bernardi, Y. Maday, and A. T. Patera. Domain decomposition by the mortar element method. *Asymptotic and Numerical Methods for Partial Differential Equations with Critical Parameters*, pages 269–286, 1993. (Cited on page 18.)

[19] A. P. S. Bhalla, R. Bale, B. E. Griffith, and N. A. Patankar. A unified mathematical framework and an adaptive numerical method for fluid-structure interaction with rigid, deforming, and elastic bodies. *Journal of Computational Physics*, 250:446–476, 2013. (Cited on page 2.)

[20] P. Birken. Termination criteria for inexact fixed-point schemes. *Numerical Linear Algebra with Applications*, 22(4):702–716, 2015. (Cited on page 31.)

[21] P. Birken, T. Gleim, D. Kuhl, and A. Meister. Fast solvers for thermal fluid structure interaction. In *ECCOMAS Marine V*, Hamburg, 2013. (Cited on page 2.)

[22] P. Birken, K. J. Quint, S. Hartmann, and A. Meister. A time-adaptive fluid-structure interaction method for thermal coupling. *Computing and Visualization in Science*, 13(2010):331–340, 2010. (Cited on pages 5 and 6.)

[23] P. Bjorstad and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 2004. (Cited on page 4.)

[24] K.-U. Bletzinger, R. Wüchner, and A. Kupzok. Algorithmic treatment of shells and free form-membranes in FSI. In H.-J. Bungartz and M. Schäfer, editors, *Fluid-Structure Interaction: Modelling, Simulation, Optimisation*, pages 336–355. Springer, 2006. (Cited on page 27.)

[25] D. Blom, T. Ertl, O. Fernandes, S. Frey, H. Klimach, V. Krupp, M. Mehl, S. Roller, D. C. Sternel, B. Uekermann, T. Winter, and A. van Zuijlen. Partitioned fluid-structure-acoustics interaction on distributed data – numerical results and visualization. In H.-J. Bungartz, P. Neumann, and E. W. Nagel, editors, *Software for Exa-scale Computing – SPPEXA 2013-2015.* Springer, 2016. (Cited on pages 107, 113, 114, 119, and 120.)

[26] D. Blom, V. Krupp, A. van Zuijlen, H. Klimach, S. Roller, and H. Bijl. On parallel scalability aspects of strongly coupled partitioned fluid-structure-acoustics interaction. In *ECCOMAS Coupled Problems*, Venice, 2015. (Cited on page 119.)

[27] D. Blom, F. Lindner, M. Mehl, K. Scheufele, B. Uekermann, and A. van Zuijlen. A review on fast quasi-newton and accelerated fixed point iterations for partitioned fluid-structure interaction simulation. In Y. Bazilevs and K. Takizava, editors, *Advances in Computational Fluid-Structure Interaction.* Springer, 2016. (Cited on page 29.)

[28] D. Blom, B. Uekermann, M. Mehl, A. van Zuijlen, and H. Bijl. Multi-level acceleration of parallel coupled partitioned fluid-structure interaction with manifold mapping. In M. Mehl, M. Bischoff, and M. Schäfer, editors, *International Workshop on Computational Engineering CE 2014.* Springer, 2015. (Cited on pages 5 and 37.)

[29] D. Blom, A. van Zuijlen, and H. Bijl. Acceleration of strongly coupled fluid-structure interaction with manifold mapping. In *WCCM XI*, Barcelona, 2014. (Cited on pages 5 and 37.)

[30] D. Blom, A. V. Zuijlen, and H. Bijl. Arbitrarily high order time integration for partitioned fluid-structure interaction simulations using integral deferred corrections. *Computational Physics*, 2016. Under Review. (Cited on page 136.)

[31] A. D. Boer, A. van Zuijlen, and H. Bijl. Comparison of conservative and consistent approaches for the coupling of non-matching meshes. *Computer Methods in Applied Mechanics and Engineering*, 197(49):4284–4297, 2008. (Cited on pages 18, 84, and 86.)

[32] A. Bogaers, S. Kok, B. Reddy, and T. Franz. Extending the robustness and efficiency of artificial compressibility for partitioned fluid-structure interactions. *Computer Methods in Applied Mechanics and Engineering*, 283:1278–1295, 2014. (Cited on page 5.)

[33] A. Bogaers, S. Kok, B. Reddy, and T. Franz. Quasi-Newton methods for implicit black-box FSI coupling. *Computer Methods in Applied Mechanics and Engineering*, 279:113–132, 2014. (Cited on pages 5, 37, and 38.)

[34] M. Brenk. *Algorithmische Aspekte der Fluid-Struktur-Wechselwirkung auf kartesischen Gittern.* PhD thesis, Universität Stuttgart, 2007. (Cited on pages 8, 13, and 18.)

[35] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, pages 577–593, 1965. (Cited on page 43.)

[36] M. Buhmann. Radial basis functions. *Acta Numerica*, 9(January 2000):1–38, 2000. (Cited on page 18.)

[37] M. Bukač, I. Yotov, and P. Zunino. An operator splitting approach for the interaction between a fluid and a multilayered poroelastic structure. *Numerical Methods for Partial Differential Equations*, 31(4):1054–1100, 2015. (Cited on page 2.)

[38] M. Bukač, S. Čanić, and B. Muha. A partitioned scheme for fluid-composite structure interaction problems. *Journal of Computational Physics*, 281:493–517, 2015. (Cited on page 5.)

[39] H. Bungartz, J. Benk, B. Gatzhammer, M. Mehl, and T. Neckel. Partitioned simulation of fluid-structure interaction on Cartesian grids. In H.-J. Bungartz, M. Mehl, and M. Schäfer, editors, *Fluid Structure Interaction II: Modelling, Simulation, Optimization.* Springer, 2010. (Cited on page 13.)

[40] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:147–269, 2004. (Cited on page 130.)

[41] H.-J. Bungartz, H. Klimach, V. Krupp, F. Lindner, M. Mehl, S. Roller, and B. Uekermann. Fluid-acoustics interaction on massively parallel systems. In M. Mehl, M. Bischoff, and M. Schäfer, editors, *International Workshop on Computational Engineering CE 2014*. Springer, 2015. (Cited on pages 27, 70, and 99.)

[42] H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, and B. Uekermann. preCICE – a fully parallel library for multi-physics surface coupling. *Computers & Fluids*, 2016. Accepted for publication. (Cited on pages 13, 14, and 87.)

[43] H.-J. Bungartz, F. Lindner, M. Mehl, K. Scheufele, A. Shukaev, and B. Uekermann. Partitioned fluid-structure-acoustics interaction on distributed data – coupling via preCICE. In H.-J. Bungartz, P. Neumann, and E. W. Nagel, editors, *Software for Exa-scale Computing – SPPEXA 2013-2015*. Springer, 2016. (Cited on pages 70, 71, 78, 84, 87, 89, 92, 94, and 95.)

[44] H.-J. Bungartz, F. Lindner, M. Mehl, and B. Uekermann. A plug-and-play coupling approach for parallel multi-field simulations. *Computational Mechanics*, 55(6):1119–1129, 2015. (Cited on pages 20, 29, 62, and 64.)

[45] H. J. Bungartz, M. Mehl, T. Neckel, and T. Weinzierl. The PDE framework Peano applied to fluid dynamics: An efficient implementation of a parallel multiscale fluid dynamics solver on octree-like adaptive Cartesian grids. *Computational Mechanics*, 46:103–114, 2010. (Cited on page 28.)

[46] H.-J. Bungartz, M. Mehl, and M. Schäfer. *Fluid Structure Interaction II: Modelling, Simulation, Optimization.* Springer, 2010. (Cited on page 5.)

[47] E. Burman and M. A. Fernández. Stabilization of explicit coupling in fluid-structure interaction involving fluid incompressibility. *Computer Methods in Applied Mechanics and Engineering*, 198(5-8):766–784, 2009. (Cited on page 5.)

[48] P. Cardiff, A. Karač, and A. Ivanković. A large strain finite volume method for orthotropic bodies with general material orientations. *Computer Methods in Applied Mechanics and Engineering*, 268:318–335, 2014. (Cited on page 27.)

[49] E. Casoni, G. Houzeaux, and M. Vázquez. Parallel aspects of fluid-structure interaction. *Procedia Engineering*, 61:117–121, 2013. (Cited on page 7.)

[50] P. Causin, J. Gerbeau, and F. Nobile. Added-mass effect in the design of partitioned algorithms for fluid-structure problems. Rapport de recherche, INRIA, 2005. (Cited on page 5.)

[51] J. R. Cebral and R. Loehner. Conservative load projection and tracking for fluid-structure problems. *AIAA Journal*, 35(4):687–692, 1997. (Cited on pages 1 and 18.)

[52] P. Chen. Uncertainty quantification in multiphysical and multiscale modelling of cardiovascular system. 2013. (Cited on page 127.)

[53] J.-F. Cori, S. Etienne, A. Garon, and D. Pelletier. High-order implicit Runge-Kutta time integrators for fluid-structure interactions. *International Journal for Numerical Methods in Fluids*, 2015. (Cited on page 6.)

[54] P. Crosetto, S. Deparis, G. Fourestey, and A. Quarteroni. Parallel algorithms for fluid-structure interaction problems in haemodynamics. *SIAM Journal on Scientific Computing*, 33(4):1598–1622, 2011. (Cited on pages 3 and 5.)

[55] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. Stewart. Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization. *Mathematics of Computation*, 30(136):772–795, 1976. (Cited on page 92.)

[56] A. de Boer, M. S. van der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers & Structures*, 85:784–795, 2007. (Cited on page 19.)

[57] G. de Nayer and M. Breuer. Fluid-structure interaction of thin structures in turbulent flows. In S. Wagner, A. Bode, H. Satzger, and M. Brehm, editors, *High Performance Computing in Science and Engineering, Garching/Munich*, pages 116–117. Springer, 2014. (Cited on pages 2, 3, and 103.)

[58] G. De Nayer, A. Kalmbach, M. Breuer, S. Sicklinger, and R. Wüchner. Flow past a cylinder with a flexible splitter plate: A complementary experimental-numerical investigation and a new FSI test case (FSI-PfS-1a). *Computers & Fluids*, 99:18–43, 2014. (Cited on pages 2, 3, 11, 70, 98, 101, 102, and 105.)

[59] J. De Ridder, J. Degroote, K. Van Tichelen, P. Schuurmans, and J. Vierendeels. Predicting turbulence-induced vibration in axial annular flow by means of large-eddy simulations. *Journal of Fluids and Structures*, 61:115–131, 2016. (Cited on page 123.)

[60] J. Degroote, K.-J. Bathe, and J. Vierendeels. Performance of a new partitioned procedure versus a monolithic procedure in fluid-structure interaction. *Computers & Structures*, 87(11-12):793–801, 2009. (Cited on pages 5, 6, 8, 19, 32, 34, 37, 38, and 135.)

[61] J. Degroote, J. Bols, and L. Taelman. Comparison between two different decompositions for the solution of fluid-structure interaction problems. In *ECCOMAS Coupled Problems V*, Ibiza, 2013. (Cited on page 5.)

[62] J. Degroote, P. Bruggeman, R. Haelterman, and J. Vierendeels. Stability of a coupling technique for partitioned solvers in FSI applications. *Computers & Structures*, 86(23-24):2224–2234, 2008. (Cited on pages 38 and 39.)

[63] J. Degroote, R. Haelterman, S. Annerel, P. Bruggeman, and J. Vierendeels. Performance of partitioned procedures in fluid-structure interaction. *Computers & Structures*, 88(7-8):446–457, 2010. (Cited on pages 36, 37, and 39.)

[64] J. Degroote, A. Swillens, P. Bruggeman, R. Haelterman, P. Segers, and J. Vierendeels. Simulation of fluid-structure interaction with the interface artificial compressibility method. *International Journal for Numerical Methods in Biomedical Engineering*, 26(3-4):276–289, 2010. (Cited on page 5.)

[65] J. Degroote and J. Vierendeels. Multi-solver algorithms for the partitioned simulation of fluid-structure interaction. *Computer Methods in Applied Mechanics and Engineering*, 200:2195–2210, 2011. (Cited on pages 5 and 37.)

[66] S. Deparis, M. Discacciati, G. Fourestey, and A. Quarteroni. Fluid-structure algorithms based on Steklov-Poincaré operators. *Computer Methods in Applied Mechanics and Engineering*, 195:5797–5812, 2006. (Cited on pages 5, 8, 30, and 37.)

[67] S. Deparis, D. Forti, and A. Quarteroni. A Rescaled Localized Radial Basis Function Interpolation on Non-Cartesian and Nonconforming Grids. *SIAM Journal on Scientific Computing*, 36(6):A2745–A2762, 2014. (Cited on pages 19 and 138.)

[68] W. Dettmer and D. Perić. A computational framework for fluid-structure interaction: Finite element formulation and applications. *Computer Methods in Applied Mechanics and Engineering*, 195:5754–5779, 2006. (Cited on page 5.)

[69] W. Dettmer and D. Perić. A new staggered scheme for fluid-structure interaction. *International Journal for Numerical Methods in Engineering*, 93(1):1–22, 2013. (Cited on page 5.)

[70] Y. Diekmann. Möglichkeiten der Kopplung von Fluid- und Struktursimulationen. Semesterarbeit, Technische Universität München, 2015. (Cited on page 123.)

[71] J. Dongarra, J. Hittinger, J. Bell, L. Chacón, R. Falgout, M. Heroux, P. Hovland, E. Ng, C. Webster, and S. Wild. Applied mathematics research for exascale computing. Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2014. (Cited on page 137.)

[72] M. Dörfel and B. Simeon. Fluid-structure interaction: Acceleration of strong coupling by preconditioning of the fixed-point iteration. In A. Angiani, R. L. Davidchack, E. Georgoulis, A. N. Gorban, J. Levesley, and M. V. Tretyakov, editors, *Numerical Mathematics and Advanced Applications 2011*, pages 741–749. Springer Berlin Heidelberg, 2011. (Cited on page 5.)

[73] F. Duchaine, S. Jauré, D. Poitou, E. Quémerais, G. Staffelbach, T. Morel, and L. Gicquel. Analysis of high performance conjugate heat transfer with the OpenPALM coupler. *Parallel Computing*, 2013. (Cited on pages 25 and 137.)

[74] T. Dunne and R. Rannacher. Adaptive finite element approximation of fluid-structure interaction based on an Eulerian variational formulation. In H.-J. Bungartz and M. Schäfer, editors, *Fluid Structure Interaction I: Modelling, Simulation, Optimization*. Springer, 2006. (Cited on page 6.)

[75] T. Dunne, R. Rannacher, and T. Richter. Numerical simulation of fluid-structure interaction based on monolithic variational formulations. *Fundamental Trends in Fluid-Structure Interaction*, 1:1–75, 2010. (Cited on page 6.)

[76] A. Dutt, L. Greengard, and V. Rokhlin. Spectral deferred correction methods for ordinary differential equations. *BIT Numerical Mathematics*, 40(2):241–266, 2000. (Cited on page 136.)

[77] S. C. Eisenstat and H. F. Walker. Choosing the forcing terms in an inexact newton method. *SIAM Journal on Scientific Computing*, 17(1):16–32, 1996. (Cited on page 31.)

[78] H. R. Fang and Y. Saad. Two classes of multisecant methods for nonlinear acceleration. *Numerical Linear Algebra with Applications*, 16:197–221, 2009. (Cited on pages 19, 32, 34, 35, 37, 48, 91, and 135.)

[79] I.-G. Farcas. High Dimensional Uncertainty Quantification of Fluid-Structure Interaction. Master's thesis, Technische Universität München, 2015. (Cited on pages 108 and 132.)

[80] I.-G. Farcas, B. Uekermann, T. Neckel, and H.-J. Bungartz. Non-intrusive uncertainty analysis of fluid-structure interaction with adaptive sparse grid collocation and polynomial chaos expansion. *SIAM Journal on Scientific Computing*, 2016. In preparation. (Cited on pages 108, 128, 130, 131, 133, and 134.)

[81] C. Farhat, P. Geuzaine, and G. Brown. Application of a three-field nonlinear fluid-structure formulation to the prediction of the aeroelastic parameters of an F-16 fighter. *Computers & Fluids*, 32:3–29, 2003. (Cited on page 1.)

[82] C. Farhat and V. K. Lakshminarayan. An ALE formulation of embedded boundary methods for tracking boundary layers in turbulent fluid-structure interaction problems. *Journal of Computational Physics*, 263:53–70, 2014. (Cited on page 6.)

[83] C. Farhat and M. Lesoinne. Two efficient staggered algorithms for the serial and parallel solution of three-dimensional nonlinear transient aeroelastic problems. *Computer Methods in Applied Mechanics and Engineering*, 182:499–515, 2000. (Cited on pages 4, 5, 8, and 19.)

[84] C. Farhat, M. Lesoinnea, and P. Letallec. Load and motion transfer algorithms for fluid-structure interaction problems with non-matching discrete interfaces: Momentum and energy conservation, optimal discretization and application to aeroelasticity. *Computer Methods in Applied Mechanics and Engineering*, 7825(97):95–114, 1998. (Cited on page 18.)

[85] C. Farhat, K. G. van der Zee, and P. Geuzaine. Provably second-order time-accurate loosely-coupled solution algorithms for transient nonlinear computational aeroelasticity. *Computer Methods in Applied Mechanics and Engineering*, 195:1973–2001, 2006. (Cited on page 5.)

[86] C. Felippa, K. Park, and C. Farhat. Partitioned analysis of coupled mechanical systems. *Engineering Computations*, (5):123–133, 2001. (Cited on pages 5 and 8.)

[87] C. A. Felippa and T. L. Geers. Partitioned analysis for coupled mechanical systems. *Engineering Computations*, 5(2):123–133, 1988. (Cited on page 4.)

[88] C. A. Felippa, K. C. Park, and M. R. Ross. A classification of interface treatments for FSI. In H.-J. Bungartz, M. Mehl, and M. Schäfer, editors, *Fluid Structure Interaction II: Modelling, Simulation, Optimization*, pages 27–51. Springer, 2010. (Cited on page 18.)

[89] M. A. Fernández, J.-F. Gerbeau, and C. Grandmont. A projection semi-implicit scheme for the coupling of an elastic structure with an incompressible fluid. *International Journal for Numerical Methods in Engineering*, 69(4):794–821, 2007. (Cited on page 5.)

[90] M. A. Fernández, M. Landajuela, J. Mullaert, and M. Vidrascu. Robin-Neumann schemes for incompressible fluid-structure interaction. In *Domain Decomposition Methods in Science and Engineering XXII*, Lugano, 2016. (Cited on page 5.)

[91] M. A. Fernández and M. Moubachir. A Newton method using exact Jacobians for solving fluid-structure coupling. *Computers & Structures*, 83:127–142, 2005. (Cited on pages 5 and 56.)

[92] M. Fischer, M. Firl, H. Masching, and K. Bletzinger. Optimization of non-linear structures based on object-oriented parallel programming. In *ECT 7*, Stirlingshire, UK, 2010. (Cited on page 27.)

[93] L. Formaggia, J.-F. Gerbeau, F. Nobile, and A. Quarteroni. On the coupling of 3D and 1D Navier–Stokes equations for flow problems in compliant vessels. *Computer Methods in Applied Mechanics and Engineering*, 191(6):561–582, 2001. (Cited on page 56.)

[94] C. Förster, W. A. Wall, and E. Ramm. Artificial added mass instabilities in sequential staggered coupling of nonlinear structures and incompressible viscous flows. *Computer Methods in Applied Mechanics and Engineering*, 196(7):1278–1293, 2007. (Cited on page 5.)

[95] D. Forti. *Parallel Algorithms for the Solution of Large-Scale Fluid-Structure Interaction Problems in Hemodynamics.* PhD thesis, École Polytechnique Fédérale de Lausanne, 2016. (Cited on page 3.)

[96] F. Franzelin, P. Diehl, and D. Pflüger. Non-intrusive uncertainty quantification with sparse grids for multivariate peridynamic simulations. In *Meshfree Methods for Partial Differential Equations VII*, pages 115–143. Springer, 2015. (Cited on pages 130 and 131.)

[97] V. Ganine, N. Hills, and B. Lapworth. Nonlinear acceleration of coupled fluid-structure transient thermal problems by Anderson mixing. *International Journal for Numerical Methods in Fluids*, 2012. (Cited on page 34.)

[98] D. Gardner, C. Woodward, D. Reynolds, G. Hommes, S. Aubry, and A. Arsenlis. Implicit integration methods for dislocation dynamics. *Modelling and Simulation in Materials Science and Engineering*, 23(2):025006, 2015. (Cited on page 34.)

[99] B. Gatzhammer. *Efficient and Flexible Partitioned Simulation of Fluid-Structure Interactions.* PhD thesis, Technische Universität München, 2015. (Cited on pages 8, 9, 10, 11, 13, 18, 20, 22, 23, 24, 28, 36, 38, 44, 45, 48, 55, 63, 72, 74, 78, 80, 82, 83, 85, 86, 87, 91, 135, and 136.)

[100] M. Gee, U. Küttler, and W. Wall. Truly monolithic algebraic multigrid for fluid-structure interaction. *International Journal for Numerical Methods in Engineering*, 2011. (Cited on page 5.)

[101] J.-F. Gerbeau and M. Vidrascu. A quasi-Newton algorithm based on a reduced model for fluid-structure interaction problems in blood flows. *ESAIM: Mathematical Modelling and Numerical Analysis*, 37(4):631–647, 2003. (Cited on page 5.)

[102] M. Glück, M. Breuer, F. Durst, A. Halfmann, and E. Rank. Computation of fluid-structure interaction on lightweight structures. *Journal of Wind Engineering and Industrial Aerodynamics*, 89:1351–1368, 2001. (Cited on pages 1 and 5.)

[103] G. H. Golub and C. F. Van Loan. *Matrix Computations*, volume 3. JHU Press, 2012. (Cited on page 35.)

[104] R. S. Gorla, S. S. Pai, and J. J. Rusick. Probabilistic study of fluid-structure interaction. In *ASME Turbo Expo*, 2002. (Cited on page 128.)

[105] M. Graczyk and T. Moan. A probabilistic assessment of design sloshing pressure time histories in LNG tanks. *Ocean Engineering*, 35, 2008. (Cited on page 2.)

[106] P. M. Gresho and R. L. Sani. *Incompressible Flow and the Finite Element Method. Volume 1: Advection-Diffusion and Isothermal Laminar Flow.* John Wiley and Sons, Inc., 1998. (Cited on page 11.)

[107] D. Groen, S. Zasada, and P. Coveney. Survey of multiscale and multiphysics applications and communities. *Computing in Science Engineering*, 16(2):34–43, 2014. (Cited on page 22.)

[108] R. Haelterman. *Analytical Study of the Least Squares Quasi-Newton Method for Interaction Problems.* PhD thesis, Ghent University, 2009. (Cited on page 37.)

[109] R. Haelterman, A. Bogaers, B. Uekermann, K. Scheufele, and M. Mehl. Improving the performance of the partitioned QN-ILS procedure for fluid-structure interaction problems: filtering. *Computers & Structures*, 171:9–17, 2016. (Cited on pages 29, 48, 49, and 58.)

[110] R. Haelterman, J. Degroote, D. van Heule, and J. Vierendeels. The quasi-newton least squares method: A new and fast secant method analyzed for linear systems. *SIAM Journal on Numerical Analysis*, 47(3):2347–2368, 2009. (Cited on pages 35, 37, and 38.)

[111] M. Heil, A. L. Hazel, and J. Boyle. Solvers for large-displacement fluid-structure interaction problems: Segregated versus monolithic approaches. *Computational Mechanics*, 43:91–101, 2008. (Cited on page 6.)

[112] S. Herb. Development of a FEM Code for Fluid-Structure Coupling. Master's thesis, University of Stuttgart, 2015. (Cited on page 28.)

[113] G. Houzeaux, R. Aubry, and M. Vázquez. Extension of fractional step techniques for incompressible flows: The preconditioned Orthomin(1) for the pressure Schur complement. *Computers & Fluids*, 44(1):297–313, 2011. (Cited on page 26.)

[114] G. Houzeaux and J. Principe. A variational subgrid scale model for transient incompressible flows. *International Journal of Computational Fluid Dynamics*, 22(3):135–152, 2008. (Cited on pages 26 and 102.)

[115] G. Houzeaux, M. Vázquez, R. Aubry, and J. Cela. A massively parallel fractional step solver for incompressible flows. *Journal of Computational Physics*, 228(17):6316–6332, 2009. (Cited on page 26.)

[116] T. J. Hughes. Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods. *Computer methods in applied mechanics and engineering*, 127(1):387–401, 1995. (Cited on page 26.)

[117] S. R. Idelsohn, J. Marti, A. Limache, and E. Oñate. Unified Lagrangian formulation for elastic solids and incompressible fluids: Application to fluid-structure interaction problems via the PFEM. *Computer Methods in Applied Mechanics and Engineering*, 197:1762–1776, 2008. (Cited on page 6.)

[118] B. Irons and R. Tuck. A version of the Aitken accelerator for computer iteration. *International Journal for Numerical Methods in Engineering*, 1:275–277, 1969. (Cited on page 32.)

[119] W. Joppich and M. Kürschner. MpCCI – A tool for the simulation of coupled applications. *Concurrency Computation Practice and Experience*, 18:183–192, 2006. (Cited on page 25.)

[120] D. Kamensky, M.-C. Hsu, D. Schillinger, J. A. Evans, A. Aggarwal, M. S. Sacks, and T. J. R. Hughes. A variational immersed boundary framework for fluid-structure interaction : Isogeometric implementation and application to bioprosthetic heart valves. *Computer Methods in Applied Mechanics and Engineering*, 2014. In review. (Cited on page 2.)

[121] S. Kataoka, S. Minami, H. Kawai, T. Yamada, and S. Yoshimura. A parallel iterative partitioned coupling analysis system for large-scale acoustic fluid-structure interactions. *Computational Mechanics*, 2014. (Cited on pages 3, 8, and 24.)

[122] D. Keyes, L. C. McInnes, C. S. Woodward, W. Gropp, E. Myra, M. Pernice, J. Bell, J. Brown, A. Clo, J. Connors, E. Constantinescu, D. Estep, K. Evans, C. Farhat, A. Hakim, G. Hammond, G. Hansen, J. Hill, T. Isaac, X. Jiao, K. Jordan, D. Kaushik, E. Kaxiras, A. Koniges, K. Lee, A. Lott, Q. Lu, J. Magerlein, R. Maxwell, M. McCourt, M. Mehl, R. Pawloski, A. Randles, D. Reynolds, B. Riviere, U. Rüde, T. Scheibe, J. Shadid, B. Sheehan, M. Shephard, A. Siegel, B. Smith, X. Tang, C. Wilson, and B. Wohlmuth. Multiphysics simulations: Challenges and opportunities. *High Performance Computing Applications*, 27(1):4–83, 2012. (Cited on pages 1 and 31.)

[123] H. Klimach, K. Jain, and S. Roller. End-to-end parallel simulations with APES. In *PARCO*, Munich, 2013. (Cited on page 27.)

[124] T. Klöppel, A. Popp, U. Küttler, and W. A. Wall. Fluid–structure interaction for non-conforming interfaces based on a dual mortar formulation. *Computer Methods in Applied Mechanics and Engineering*, 200(45):3111–3126, 2011. (Cited on page 18.)

[125] F. Kong and X.-C. Cai. Scalability study of an implicit solver for coupled fluid-structure interaction problems on unstructured meshes in 3D. *International Journal of High Performance Computing Applications*, pages 1–13, 2016. (Cited on pages 3 and 5.)

[126] M. Kornhaas, M. Schäfer, and D. C. Sternel. Efficient numerical simulation of aeroacoustics for low mach number flows interacting with structures. *Computational Mechanics*, 55(6):1143–1154, 2015. (Cited on page 28.)

[127] P. Kuberry and H. Lee. A decoupling algorithm for fluid-structure interaction problems based on optimization. *Computer Methods in Applied Mechanics and Engineering*, 267(October):594–605, 2013. (Cited on pages 5 and 37.)

[128] U. Küttler and W. A. Wall. Fixed-point fluid-structure interaction solvers with dynamic relaxation. *Computational Mechanics*, 43(1):61–72, 2008. (Cited on pages 5, 19, and 32.)

[129] F. Lindner, M. Mehl, K. Scheufele, and B. Uekermann. A comparison of various quasi-Newton schemes for partitioned fluid-structure interaction. In *ECCOMAS Coupled Problems*, Venice, 2015. (Cited on pages 5, 19, 29, and 37.)

[130] G. Link, M. Kaltenbacher, M. Breuer, and M. Döllinger. A 2D finite-element scheme for fluid-solid-acoustic interactions and its application to human phonation. *Computer Methods in Applied Mechanics and Engineering*, 198(41-44):3321–3334, 2009. (Cited on pages 2 and 113.)

[131] L. Liu and D. E. Keyes. Field-split preconditioned inexact Newton algorithms. *SIAM Journal on Scientific Computing*, 37(3):1388–1409, 2015. (Cited on page 8.)

[132] A. Loeven, J. Witteveen, and H. Bijl. Efficient uncertainty quantification using a two-step approach with chaos collocation. In *ECCOMAS CFD 2006*, Egmond aan Zee, The Netherlands, 2006. (Cited on page 128.)

[133] J. Loffeld and C. Woodward. Considerations and the implementation and use of Anderson acceleration on parallel computers. In *Advances in the Mathematical Sciences: Research from the 2015 Association for Women in Mathematics Symposium*, 2016. To appear. (Cited on page 92.)

[134] C. Lothode, M. Durand, A. Leroyer, M. Visonneau, Y. Roux, and L. Dorez. Fluid-structure interaction analysis of an hydrofoil. In *ECCOMAS Marine V*, Hamburg, 2013. (Cited on page 2.)

[135] P. A. Lott, H. F. Walker, C. S. Woodward, and U. M. Yang. An accelerated Picard method for nonlinear systems related to variably saturated flow. *Advances in Water Resources*, 38:92–101, 2012. (Cited on pages 34, 36, and 37.)

[136] N. Maman and C. Farhat. Matching fluid and structure meshes for aeroelastic computations: A parallel approach. *Computers & Structures*, 54(4):779–785, 1995. (Cited on pages 1 and 18.)

[137] L. Marks and D. Luke. Robust mixing for ab initio quantum mechanical calculations. *Physical Review B*, 78(7), 2008. (Cited on pages 35, 48, and 51.)

[138] J. Martorell, R. Pons, L. Dux-Santoy, J. F. Rodriguez-Paloromes, J. J. Molins, and A. Evangelista. 4D-MRI coupled to fluid dynamics simulations to improve patient management. *Tecnicas Endovasculares*, 18:25–30, 2015. (Cited on page 110.)

[139] H. G. Matthies, R. Niekamp, and J. Steindorf. Algorithms for strong coupling procedures. *Computer Methods in Applied Mechanics and Engineering*, 195(17):2028–2049, 2006. (Cited on page 22.)

[140] H. G. Matthies and J. Steindorf. Partitioned strong coupling algorithms for fluid-structure interaction. *Computers & Structures*, 81(8-11):805–812, 2003. (Cited on page 5.)

[141] M. Mayr, T. Klöppel, W. A. Wall, and M. W. Gee. A temporal consistent monolithic approach to fluid-structure interaction enabling single field predictors. *SIAM Journal on Scientific Computing*, 37(1):B30–B59, 2015. (Cited on page 6.)

[142] M. Mehl, B. Uekermann, H. Bijl, D. Blom, B. Gatzhammer, and A. van Zuijlen. Parallel coupling numerics for partitioned fluid-structure interaction simulations. *Computers and Mathematics with Applications*, (4):869–891, 2016. (Cited on pages 5, 28, 29, 37, 38, 40, 54, 55, 56, and 58.)

[143] Q. Meng and M. Berzins. Scalable large-scale fluid–structure interaction solvers in the Uintah framework via hybrid task-based parallelism algorithms. *Concurrency and Computation: Practice and Experience*, 26(7):1388–1407, 2014. (Cited on page 22.)

[144] C. Michler. An interface Newton-Krylov solver for fluid-structure interaction. *International Journal for Numerical Methods in Fluids*, 47(10-11):1189–1195, 2004. (Cited on pages 5 and 36.)

[145] C. Michler, S. J. Hulshoff, E. H. van Brummelen, and R. de Borst. A monolithic approach to fluid-structure interaction. *Computers & Fluids*, 33:839–848, 2004. (Cited on page 6.)

[146] C. Michler, E. H. van Brummelen, and R. de Borst. An investigation of Interface-GMRES(R) for fluid-structure interaction problems with flutter and divergence. *Computational Mechanics*, 47:17–29, 2011. (Cited on pages 5 and 36.)

[147] V. Mikerov. A Fixed-Grid Flow Solver for Fluid-Structure Interaction with the Coupling Library preCICE. Master's thesis, Technische Universität München, 2015. (Cited on page 28.)

[148] S. Minami and S. Yoshimura. Performance evaluation of nonlinear algorithms with line-search for partitioned coupling techniques for fluid-structure interactions. *International Journal for Numerical Methods in Fluids*, 2010. (Cited on pages 5 and 37.)

[149] D. Mira, M. Zavala-Ake, M. Avila, H. Owen, J. C. Cajas, M. Vazquez, and G. Houzeaux. Heat transfer effects on a fully premixed methane impinging flame. *Flow, Turbulence and Combustion*, 97(1):339–361, 2016. (Cited on page 2.)

[150] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annual Review of Fluid Mechanics*, 37:239–261, 2005. (Cited on page 6.)

[151] H. N. Najm. Uncertainty quantification and polynomial chaos techniques in computational fluid dynamics. *Annual Review of Fluid Mechanics*, 41(1):35–52, 2009. (Cited on page 128.)

[152] T. Neckel. *The PDE Framework Peano: An Environment for Efficient Flow Simulations.* PhD thesis, Technische Universität München, 2009. (Cited on page 28.)

[153] V.-T. Nguyen and B. Gatzhammer. A fluid structure interactions partitioned approach for simulations of explosive impacts on deformable structures. *International Journal of Impact Engineering*, 2015. (Cited on page 28.)

[154] P. Ni. *Anderson Acceleration of Fixed-Point Iteration with Applications to Electronic Structure Computations.* PhD thesis, Worcester Polytechnic Institute, 2009. (Cited on page 34.)

[155] J. Nitsche. Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind. In *Abhandlungen aus dem mathematischen Seminar der Universität Hamburg*, volume 36, pages 9–15. Springer, 1971. (Cited on page 6.)

[156] F. Nobile, M. Pozzoli, and C. Vergara. Inexact accurate partitioned algorithms for fluid-structure interaction problems with finite elasticity in haemodynamics. *Journal of Computational Physics*, 273:598–617, 2014. (Cited on page 5.)

[157] F. Nobile and C. Vergara. An effective fluid-structure interaction formulation for vascular dynamics by generalized Robin conditions. *SIAM Journal on Scientific Computing*, 30(2):731–763, 2008. (Cited on page 5.)

[158] F. Palacios, J. Alonso, K. Duraisamy, M. Colonno, J. Hicken, A. Aranake, et al. An open-source integrated 3 computational environment for multi-physics simulation and design. In *51st AIAA Aerospace Scientific Meeting*, 2013. (Cited on page 28.)

[159] N. Parolini and M. Lombardi. Unsteady FSI simulation of downwind sails. In *ECCOMAS Marine V*, Hamburg, 2013. (Cited on page 2.)

[160] C. Peskin. Flow patterns around heart valves: a numerical method. *Journal of Computational Physics*, 271:252–271, 1972. (Cited on page 6.)

[161] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems.* PhD thesis, Technische Universität München, 2010. (Cited on page 131.)

[162] S. Piperno, C. Farhat, and B. Larrouturou. Partitioned procedures for the transient solution of coupled aroelastic problems part i: Model problem, theory and two-dimensional application. *Computer Methods in Applied Mechanics and Engineering*, 124(1):79–112, 1995. (Cited on page 1.)

[163] S. J. Plimpton, B. Hendrickson, and J. R. Stewart. A parallel rendezvous algorithm for interpolation between multiple grids. *Journal of Parallel and Distributed Computing*, 64(2):266–276, 2004. (Cited on page 137.)

[164] S. B. Pope. *Turbulent Flows.* IOP Publishing, 2001. (Cited on page 11.)

[165] P. Pulay. Convergence acceleration of iterative sequences. the case of SCF iteration. *Chemical Physics Letters*, 73(2):393–398, 1980. (Cited on page 34.)

[166] A. Purohit, A. K. Darpe, and S. Singh. A study on aerodynamic sound from an externally excited flexible structure in flow. *Computers & Fluids*, 103:100–115, 2014. (Cited on page 113.)

[167] A. Quarteroni. *Modeling the Heart and the Circulatory System*, volume 14. Springer, 2015. (Cited on page 2.)

[168] T. Richter. Goal-oriented error estimation for fluid-structure interaction problems. *Computer Methods in Applied Mechanics and Engineering*, 223-224:28–42, 2012. (Cited on page 5.)

[169] T. Rohwedder and R. Schneider. An analysis for the diis acceleration method used in quantum chemistry calculations. *Journal of Mathematical Chemistry*, 49(9):1889–1914, 2011. (Cited on pages 34, 35, and 37.)

[170] M. R. Ross, C. A. Felippa, K. Park, and M. a. Sprague. Treatment of acoustic fluid-structure interaction by localized Lagrange multipliers: Formulation. *Computer Methods in Applied Mechanics and Engineering*, 197(33-40):3057–3079, 2008. (Cited on pages 5 and 8.)

[171] M. R. Ross, M. A. Sprague, C. A. Felippa, and K. Park. Treatment of acoustic fluid-structure interaction by localized Lagrange multipliers and comparison to alternative interface-coupling methods. *Computer Methods in Applied Mechanics and Engineering*, 198(9-12):986–1005, 2009. (Cited on page 5.)

[172] A. Rusch. Extending SU2 to Fluid-Structure Interaction via preCICE. Bachelor's thesis, Technische Universität München, 2016. (Cited on pages 28, 108, 123, and 124.)

[173] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986. (Cited on page 88.)

[174] S. Sathe, R. Benney, R. Charles, E. Doucette, J. Miletti, M. Senga, K. Stein, and T. Tezduyar. Fluid-structure interaction modeling of complex parachute designs with the space-time finite element techniques. *Computers & Fluids*, 36(1):127–135, 2007. (Cited on pages 1 and 5.)

[175] F. Schäfer, S. Müller, T. Uffinger, S. Becker, J. Grabinger, and M. Kaltenbacher. Fluid-structure-acoustic interaction of the flow past a thin flexible structure. *AIAA Journal*, 48(4):738–748, 2010. (Cited on pages 2, 5, and 113.)

[176] K. Scheufele. Robust Quasi-Newton Methods for Partitioned Fluid-Structure Simulations. Master's thesis, University of Stuttgart, 2015. (Cited on pages 29, 34, 36, 37, 38, 39, 58, and 61.)

[177] K. Scheufele and M. Mehl. Multi-secant quasi-Newton variants for parallel fluid-structure simulations – and other multi-physics applications. *SIAM Journal on Scientific Computing, Special Issue Copper Mountain*, 2016. In preparation. (Cited on pages 96, 97, and 138.)

[178] T. Scholcz, A. van Zuijlen, and H. Bijl. Space-mapping in fluid-structure interaction problems. *Computer Methods in Applied Mechanics and Engineering*, 281:162–183, 2014. (Cited on pages 5 and 37.)

[179] S. Schulte. *Modulare und hierarchische Simulation gekoppelter Probleme.* PhD thesis, Technische Universität München, 1998. (Cited on page 1.)

[180] J. P. Sheldon, S. T. Miller, and J. S. Pitt. Methodology for comparing coupling algorithms for fluid-structure interaction problems. *World Journal of Mechanics*, 4(02):54, 2014. (Cited on pages 6 and 61.)

[181] A. K. Shukaev. A Fully Parallel Process-to-Process Intercommunication Technique for preCICE. Master's thesis, Technische Universität München, 2015. (Cited on pages 10, 24, 70, 78, 82, and 100.)

[182] S. Sicklinger. *Stabilized Co-Simulation of Coupled Problems Including Fields and Signals.* PhD thesis, Technischer Universität München, 2014. (Cited on pages 2, 25, and 29.)

[183] S. Sicklinger, V. Belsky, B. Engelmann, H. Elmqvist, H. Olsson, R. Wüchner, and K.-U. Bletzinger. Interface Jacobian-based co-simulation. *International Journal for Numerical Methods in Engineering*, 98(6):418–444, 2014. (Cited on page 5.)

[184] S. Slattery, P. Wilson, and R. Pawlowski. The data transfer kit: A geometric rendezvous-based tool for multiphysics data transfer. In *M & C*, Sun Valley, Idaho, 2013. (Cited on pages 24 and 137.)

[185] M. J. Smith, C. E. S. Cesnik, and D. H. Hodges. Evaluation of algorithms suitable for data transfer between noncontiguous meshes. *ASCE Journal of Aerospace Engineering*, 13(2):52–58, 2000. (Cited on pages 18, 86, and 87.)

[186] R. C. Smith. *Uncertainty Quantification: Theory, Implementation, and Applications.* SIAM, 2013. (Cited on pages 128 and 129.)

[187] M. D. Song, E. Lefrançois, and M. Rachik. A partitioned coupling scheme extended to structures interacting with high-density fluid flows. *Computers & Fluids*, 84:190–202, 2013. (Cited on page 5.)

[188] P. Spalart and S. Allmaras. A one-equation turbulence model for aerodynamic flows. *AIAA Journal*, 94, 1992. (Cited on page 123.)

[189] K. R. Stein, R. J. Benney, V. Kalro, A. A. Johnson, and T. E. Tezduyar. Parallel computation of parachute fluid-structure interactions. *UMSI Research Report, University of Minnesota*, 97:54, 1997. (Cited on page 1.)

[190] K. R. Stein, T. E. Tezduyar, S. Sathe, and M. Senga. Simulation of parachute descent and maneuvers. In *Conference on Computation of Shell and Spatial Structures*, 2005. (Cited on page 1.)

[191] J. R. Stewart and H. C. Edwards. The SIERRA framework for developing advanced parallel mechanics applications. In *Large-Scale PDE-Constrained Optimization*, pages 301–315. Springer, 2003. (Cited on page 22.)

[192] T. Tezduyar, S. Sathe, T. Cragin, B. Nanna, B. S. Conklin, J. Pauseweg, and M. Schwaab. Modelling of fluid-structure interactions with the space-time finite elements: Arterial fluid mechanics. *International Journal for Numerical Methods in Fluids*, 54(6-8):901–922, 2007. (Cited on page 2.)

[193] T. E. Tezduyar, K. Takizawa, C. Moorman, S. Wright, and J. Christopher. Space-time finite element computation of complex fluid-structure interactions. *International Journal for Numerical Methods in Fluids*, 2010. (Cited on page 5.)

[194] F.-B. Tian, H. Dai, H. Luo, J. F. Doyle, and B. Rousseau. Fluid-structure interaction involving large deformations: 3D simulations and applications to biological systems. *Journal of Computational Physics*, 258:451–469, 2014. (Cited on pages 2 and 5.)

[195] C. E. Torres and L. A. Barba. Fast radial basis function interpolation with Gaussians by localization and iteration. *Journal of Computational Physics*, 228(14):4976–4999, 2009. (Cited on pages 19 and 138.)

[196] A. Toth and C. Kelley. Convergence analysis for Anderson acceleration. *SIAM Journal on Numerical Analysis*, pages 1–15, 2015. (Cited on pages 34 and 35.)

[197] A. Toth, C. Kelley, and R. Pawlowski. Anderson acceleration for Tiamat. In *CASL Summer Student Workshop poster*, 2015. (Cited on page 29.)

[198] A. Toth, C. Kelley, S. Slattery, S. Hamilton, K. Clarno, and R. Pawlowski. Analysis of Anderson acceleration on a simplified neutronics/thermal hydraulics system. In *Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA), and the Monte Carlo (MC) Method*, 2015. (Cited on page 34.)

[199] S. Turek and J. Hron. Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow. In H.-J. Bungartz, M. Mehl, and M. Schäfer, editors, *Fluid Structure Interaction II: Modelling, Simulation, Optimization.* Springer, 2006. (Cited on pages 54, 55, 56, 102, and 108.)

[200] S. Turek, J. Hron, M. Madlik, M. Razzaq, H. Wobker, and J. Acker. Numerical simulation and benchmarking of a monolithic multigrid solver for fluid-structure interaction problems with application to hemodynamics. In H.-J. Bungartz, M. Mehl, and M. Schäfer, editors, *Fluid Structure Interaction II: Modelling, Simulation, Optimization*, page 432. Springer Berlin Heidelberg, 2010. (Cited on page 5.)

[201] B. Uekermann. Detached-eddy simulation for the parallel finite element framework alya. Diploma thesis, Technische Universität München, 2012. (Cited on page 26.)

[202] B. Uekermann, H.-J. Bungartz, B. Gatzhammer, and M. Mehl. A parallel, black-box coupling algorithm for fluid-structure interaction. In *ECCOMAS Coupled Problems*, Ibiza, 2013. (Cited on pages 5, 29, 37, 38, and 40.)

[203] B. Uekermann, J. C. Cajas, B. Gatzhammer, G. Houzeaux, M. Mehl, and M. Vazquez. Towards fluid-structure interaction on massively parallel systems. In *WCCM XI*, Barcelona, 2014. (Cited on page 26.)

[204] B. Uekermann, B. Gatzhammer, and M. Mehl. Coupling algorithms for partitioned multi-physics simulations. In *44. Jahrestagung der Gesellschaft für Informatik*, Stuttgart, 2014. (Cited on pages 29 and 62.)

[205] S. Valcke, T. Craig, and L. Coquart. OASIS3-MCT user guide OASIS3-MCT 2.0. Technical Report 1875, CERFACS/CNRS SUC URA, 2013. (Cited on page 25.)

[206] E. van Brummelen. Partitioned iterative solution methods for fluid–structure interaction. *International Journal for Numerical Methods in Fluids*, 65(1-3):3–27, 2011. (Cited on page 37.)

[207] E. van Brummelen, C. Michler, and R. de Borst. Interface-gmres (r) acceleration of subiteration for fluid-structure-interaction problems. Technical report, Delft Aerospace Computational Science, 2005. (Cited on pages 36 and 37.)

[208] E. H. van Brummelen. Added mass effects of compressible and incompressible flows in fluid-structure interaction. *Journal of Applied Mechanics*, 76(2):1–7, 2009. (Cited on page 5.)

[209] E. H. van Brummelen, K. G. van der Zee, and R. de Borst. Space/time multigrid for a fluid-structure-interaction problem. *Applied Numerical Mathematics*, 58(May):1951–1971, 2008. (Cited on pages 5 and 37.)

[210] T. van Opstal, E. van Brummelen, and G. van Zwieten. A finite-element/boundary-element method for three-dimensional, large-displacement fluid-structure-interaction. *Computer Methods in Applied Mechanics and Engineering*, 284:637–663, 2015. (Cited on page 1.)

[211] A. van Zuijlen, A. de Boer, and H. Bijl. Higher-order time integration through smooth mesh deformation for 3D fluid-structure interaction simulations. *Journal of Computational Physics*, 224(1):414–430, 2007. (Cited on page 6.)

[212] M. Vázquez, R. Arís, J. Aguado-Sierra, G. Houzeaux, A. Santiago, M. López, P. Córdoba, M. Rivero, and J. Cajas. Alya Red CCM: HPC-based cardiac computational modelling. In *Selected Topics of Computational and Experimental Fluid Mechanics*, pages 189–207. Springer, 2015. (Cited on page 2.)

[213] M. Vázquez, G. Houzeaux, S. Koric, A. Artigues, J. Aguado-Sierra, R. Arís, D. Mira, H. Calmet, F. Cucchietti, H. Owen, et al. Alya: Multiphysics engineering simulation toward exascale. *Journal of Computational Science*, 2016. (Cited on pages 4 and 26.)

[214] C. V. Verhoosel, T. P. Scholcz, S. J. Hulshoff, and M. A. Gutiérrez. Uncertainty and reliability analysis of fluid-structure stability boundaries. *AIAA Journal*, 47(1):91–104, 2009. (Cited on page 127.)

[215] J. Vierendeels, J. Degroote, S. Annerel, and R. Haelterman. Stability issues in partitioned FSI calculations. In H.-J. Bungartz, M. Mehl, and M. Schäfer, editors, *Fluid Structure Interaction II: Modelling, Simulation, Optimization*, pages 83–102. Springer, 2010. (Cited on page 37.)

[216] J. Vierendeels, L. Lanoye, J. Degroote, and P. Verdonck. Implicit coupling of partitioned fluid-structure interaction problems with reduced order models. *Computers & Structures*, 85(11-14):970–976, 2007. (Cited on pages 5, 30, 36, 37, and 38.)

[217] H. F. Walker and P. Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(1965):1715–1735, 2011. (Cited on pages 34, 35, 36, 37, 49, and 91.)

[218] W. A. Wall. *Fluid-Struktur-Interaktion mit stabilisierten Finiten Elementen.* PhD thesis, Universität Stuttgart, 1999. (Cited on pages 1, 32, and 57.)

[219] T. Wang, S. Sicklinger, R. Wuechner, and K.-U. Bletzinger. Concept and realization of the coupling software EMPIRE in multiphysics co-simulation. In *ECCOMAS MARINE V*, Hamburg, 2013. (Cited on page 25.)

[220] T. Wang, R. Wüchner, S. Sicklinger, and K.-U. Bletzinger. Assessment and improvement of mapping algorithms for non-matching meshes and geometries in computational FSI. *Computational Mechanics*, pages 1–24, 2016. (Cited on pages 18 and 86.)

[221] T. Wick. Fully Eulerian fluid-structure interaction for time-dependent problems. *Computer Methods in Applied Mechanics and Engineering*, 255:14–26, 2013. (Cited on page 6.)

[222] N. Wiener. The homogenous chaos. *American Journal of Mathematics*, 60:897–936, 1938. (Cited on page 128.)

[223] D. C. Wilcox. *Turbulence Modeling for CFD*, volume 2. DCW Industries La Canada, 1998. (Cited on page 11.)

[224] J. A. Witteveen, S. Sarkar, and H. Bijl. Modeling physical uncertainties in dynamic stall induced fluid–structure interaction of turbine blades using arbitrary polynomial chaos. *Computers & Structures*, 85(11):866–878, 2007. (Cited on page 128.)

[225] B. I. Wohlmuth. A mortar finite element method using dual spaces for the Lagrange multiplier. *SIAM Journal on Numerical Analysis*, 38(3):989–1012, 2000. (Cited on page 18.)

[226] K. Wolf and E. Brakkee. Coupling fluids and structures codes on MPI. In *MPI Developer's Conference*, 1996. (Cited on page 1.)

[227] J. N. Wood, G. De Nayer, S. Schmidt, and M. Breuer. Experimental investigation and large-eddy simulation of the turbulent flow past a smooth and rigid hemisphere. *Flow, Turbulence and Combustion*, pages 1–41, 2016. (Cited on pages 3, 108, and 121.)

[228] P. Wriggers. *Nonlinear Finite Element Methods*. Springer Science & Business Media, 2008. (Cited on page 11.)

[229] Y. Wu and X.-C. Cai. A fully implicit domain decomposition based ALE framework for three-dimensional fluid-structure interaction with application in blood flow computation. *Journal of Computational Physics*, 258:524–537, 2014. (Cited on pages 3 and 5.)

[230] R. Wüchner. *Computational Mechanics of Form Finding and Fluid-Structure Interaction of Membrane Structures*. PhD thesis, Technischer Universit/"at M/"unchen, 2006. (Cited on page 1.)

[231] D. Xiu. *Numerical Methods for Stochastic Computations: A Spectral Method Approach*. N.J. Princeton University Press, 2010. (Cited on pages 129 and 131.)

[232] D. Xiu and G. E. Karniadakis. The Wiener-Askey polynomial chaos for stochastic differential equations. *SIAM Journal of Scientific Computing*, 24:619–644, 2002. (Cited on page 128.)

[233] D. Xiu, D. Lucor, C.-H. Su, and G. E. Karniadakis. Stochastic modeling of flow-structure interactions using generalized polynomial chaos. *Journal of Fluids Engineering*, 124(1):51–59, 2002. (Cited on page 128.)

[234] R. Yokota, L. A. Barba, and M. G. Knepley. PetRBF – A parallel O(N) algorithm for radial basis function interpolation with Gaussians. *Computer Methods in Applied Mechanics and Engineering*, 199(25-28):1793–1804, 2010. (Cited on page 19.)

[235] Y. Yu, H. Baek, and G. E. Karniadakis. Generalized fictitious methods for fluid-structure interactions: Analysis and simulations. *Journal of Computational Physics*, 2013. (Cited on page 5.)

[236] C. Yvin, A. Leroyer, and M. Visonneau. Co-simulation in fluid-structure interaction problem with rigid bodies. In *16th Numerical Towing Tank Symposium*, 2013. (Cited on page 4.)

[237] C. Zenger. Sparse grids. In W. Hackbusch, editor, *Parallel Algorithms for Partial Differential Equations, Proceedings of the Sixth GAMM-Seminar*, volume 31 of *Notes on Numerical Fluid Mechanics*, pages 241–251. Vieweg Verlag, 1991. (Cited on page 130.)

[238] J. Zudrop. *Efficient Numerical Methods for Fluid- and Electrodynamics on Massively Parallel Systems*. PhD thesis, RWTH Aachen, 2015. (Cited on pages 26 and 113.)

[239] J. Zudrop, H. Klimach, M. Hasert, K. Masilamani, and S. Roller. A fully distributed CFD framework for massively parallel systems. *Cray User Group*, 2012. (Cited on pages 4, 26, and 98.)

[240] A. H. V. Zuijlen and H. Bijl. Multi-level accelerated sub-iterations for fluid-structure interaction. In H. Bungartz, M. Mehl, and M. Schäfer, editors, *Fluid Structure Interaction II: Modelling, Simulation, Optimization*, pages 1–25. Springer, 2010. (Cited on pages 5 and 37.)