Data-Integrated Methods for Performance Improvement of Massively Parallel Coupled Simulations

Vom Stuttgarter Zentrum für Simulationswissenschaften (SC SimTech) und der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

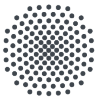
Amin Totounferoush

aus Marand, Iran

Hauptberichterin: Prof. Dr. Miriam Schulte

Mitberichter: Prof. Dr. Carlos David Pérez-Segarra

Tag der mündlichen Prüfung: 25th, July, 2022



Universität Stuttgart

Institut für Parallele und Verteilte Systeme der Universität Stuttgart

2022

Abstract

This thesis presents data-integrated methods to improve the computational performance of partitioned multi-physics simulations, particularly on highly parallel systems. Partitioned methods allow using available single-physic solvers and well-validated numerical methods for multi-physics simulations by decomposing the domain into smaller sub-domains. Each sub-domain is solved by a separate solver and an external library is incorporated to couple the solvers. This significantly reduces the software development cost and enhances the flexibility, while it introduces new challenges that must be addressed carefully. These challenges include, but are not limited to, efficient data communication between sub-domains, data mapping between not-matching meshes, inter-solver load balancing and equation coupling.

In the current work, the inter-solver communication is improved by introducing a two-level communication initialization scheme to the coupling library preCICE. The new method significantly speed ups the initialization and removes memory bottlenecks of the previous implementation. In addition, a data-driven inter-solver load balancing method is developed to efficiently distribute available computational resources between coupled single-physic solvers. This method employs both regressions and deep neural networks (DNN) for modeling the performance of the solvers and derives and solves an optimization problem to distribute the available CPU and GPU cores among solvers. To accelerate the equation coupling between strongly coupled solvers, a hybrid framework is developed that integrates DNNs and classical solvers. The DNN computes a solution estimation for each time step which is used by classical solvers as a first guess to compute the final solution. To preserve DNN's efficiency during the simulation, a dynamic re-training strategy is introduced that updates the DNN's weights on-the-fly. The cheap but accurate solution estimation by the DNN surrogate solver significantly reduces the number of subsequent classical iterations that are necessary for the solution convergence. Finally, a highly scalable simulation environment is introduced for fluid-structure interaction problems. The environment consists of highly parallel numerical solvers and an efficient and scalable coupling library. This framework is able to efficiently exploit both CPU-only and hybrid CPU-GPU machines. Numerical performance investigations using a complex test case demonstrate a very high parallel efficiency on a large number of CPUs and a significant speed-up due to the GPU acceleration.

Kurzzusammenfassung

Diese Arbeit präsentiert datenintegrierte Methoden zur Verbesserung der Rechenleistung partitionierter multiphysikalischer Simulationen, insbesondere auf hochparallelen Systemen. Partitionierte Methoden ermöglichen die Verwendung verfügbarer Einzelphysik-Löser und gut validierter numerischer Methoden für Multiphysik-Simulationen, indem das Gebiet in kleinere Teilgebiete zerlegt wird. Jedes Teilgebiet wird von einem separaten Löser gelöst, und eine externe Bibliothek ist integriert, um die Löser zu koppeln. Dies reduziert die Softwareentwicklungskosten erheblich und erhöht die Flexibilität, während es neue Herausforderungen mit sich bringt, die sorgfältig angegangen werden müssen. Zu diesen Herausforderungen gehören unter anderem eine effiziente Datenkommunikation zwischen Subdomänen, Datenmapping zwischen nicht übereinstimmenden Netzen, Lastausgleich zwischen Lösern und Gleichungskopplung.

In der vorliegenden Arbeit wird die Kommunikation zwischen Lösern verbessert, indem ein zweistufiges Kommunikations-Initialisierungsschema in die Kopplungsbibliothek preCICE eingeführt wird. Das neue Verfahren beschleunigt die Initialisierung erheblich und beseitigt Speicherengpässe der bisherigen Implementierung. Darüber hinaus wird ein datengetriebenes Lastbalancierungs-Verfahren zwischen Lösern entwickelt, um verfügbare Rechenressourcen effizient zwischen gekoppelten Einzelphysiklösern zu verteilen. Diese Methode verwendet sowohl Regressionen als auch Deep Neural Networks (DNN) zum Modellieren der Leistung der Löser, leitet ein Optimierungsproblem ab und löst es, um die verfügbaren CPU- und GPU-Kerne auf die Löser zu verteilen. Um die Gleichungskopplung zwischen stark gekoppelten Lösern zu beschleunigen, wird ein hybrides Framework entwickelt, das DNNs und klassische Löser integriert. Das DNN berechnet für jeden Zeitschritt eine Lösungsschätzung, die von klassischen Lösern als erste Schätzung zur Berechnung der endgültigen Lösung verwendet wird. Um die Effizienz des DNN während der Simulation zu erhalten, wird eine dynamische Retraing-Strategie eingeführt, die die Gewichtungen des DNN im laufenden Betrieb aktualisiert. Die billige, aber genaue Lösungsschätzung durch den DNN-Ersatzlöser reduziert die Anzahl der nachfolgenden klassischen Iterationen, die für die Lösungskonvergenz erforderlich sind, erheblich. Schließlich wird eine hochskalierbare Simulationsumgebung für Fluid-

Struktur-Interaktionsprobleme eingeführt. Die Umgebung besteht aus hochparallelen numerischen Lösern und einer effizienten und skalierbaren Kopplungsbibliothek. Dieses Framework ist in der Lage, sowohl reine CPU- als auch Hybrid-CPU-GPU-Maschinen effizient zu nutzen. Untersuchungen der numerischen Performance anhand eines komplexen Testfalls zeigen eine sehr hohe parallele Effizienz auf einer großen Anzahl von CPUs und eine deutliche Beschleunigung durch die Verwendung von GPUs.

Acknowledgment

At the end of this wonderful journey, I would like to take the opportunity to thank those who supported and contributed directly and indirectly to this thesis.

First of all, I would like to express my gratitude to my supervisor Prof. Dr. rer. nat. Miriam Schulte for her continued support, deep understanding, motivation, and insightful advice over the last five years. Thank you Miriam for creating such a positive and peaceful environment, and spending countless hours advising me. I enjoyed every moment of working with you.

I would also like to thank the CTTC lab and especially Prof. Dr. Assensi Oliva for hosting me during my research stay in Barcelona TECH. I learned a lot during my visit and enjoyed collaborating with them.

I have collaborated with many people in different phases of my doctoral studies. I would like to thank the preCICE team, specifically Dr. Benjamin Uekermann for his support and insight. I would also like to thank my previous colleague Florian whom I collaborated with substantially. I also thank my colleagues and office mates Klaudius and Kyle for the insightful discussions, mental support, and more importantly adding fun to the daily work. I would also like to thank my friends and colleagues Alireza, Theresa and Neda for the tight collaborations and all the support I received.

My deepest gratitude goes to my family and wonderful parents who always supported me, not only in my education but also in every moment of my life with their love and care. Most importantly, I express gratitude to my beloved wife, Fatemeh, who stood behind me at every tough moment. Without her support and love, this would not be possible. Thank you, Fatemeh for your understanding, patience, and support.

Publication List

The content of this dissertation is partially published in the following papers:

Journal papers:

1- **Amin Totounferoush,** Frédéric Simonis, Benjamin Uekermann, and Miriam Schulte. "Efficient and Scalable Initialization of Partitioned Coupled Simulations with preCICE." Algorithms 14, no. 6 (2021): 166.

2- **Amin Totounferoush,** Neda Ebrahimi Pour, Juri Schröder, Sabine Roller, and Miriam Mehl. "A Data-Based Inter-Code Load Balancing Method for Partitioned Solvers." Journal of Computational Science (2021): 101329.

3- Naseri, Alireza, **Amin Totounferoush,** Ignacio González, Miriam Mehl, and Carlos David Pérez-Segarra. "A scalable framework for the partitioned solution of fluid–structure interaction problems." Computational Mechanics 66 (2020): 471-489.

Peer-reviewed conference/workshop papers:

4- Lindner, Florian, **Amin Totounferoush,** Miriam Mehl, Benjamin Uekermann, Neda Ebrahimi Pour, Verena Krupp, Sabine Roller et al. "ExaFSA: Parallel Fluid-Structure-Acoustic Simulation." Software for Exascale Computing-SPPEXA 2016-2019 136 (2020): 271.

5- **Amin Totounferoush,** Neda Ebrahimi Pour, Sabine Roller, and Miriam Mehl. "Parallel Machine Learning of Partial Differential Equations", IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2021.

6-**Amin Totounferoush,** Alireza Naseri, Jorge Chiva, Assensi Oliva, and Miriam Mehl."A GPU accelerated framework for partitioned solution of fluid structure interaction." 14th World Congress on Computational Mechanics (WCCM), ECCOMAS Congress (2020).

7- **Amin Totounferoush,** Neda Ebrahimi Pour, Juri Schröder, Sabine Roller, and Miriam Mehl. "A new load balancing approach for coupled multi-physics simulations." In 2019 IEEE7- International Parallel and Distributed Processing Symposium

Workshops (IPDPSW), pp. 676-682. IEEE, 2019. **(best paper award)**

Pre-prints:

8- Chourdakis, Gerasimos, Kyle Davis, Benjamin Rodenberg, Miriam Schulte, Frédéric Simonis, Benjamin Uekermann, Georg Abrams, Hans-Joachim Bungartz, Lucia Cheung Yau, Ishaan Desai, Konrad Eder, Richard Hertrich, Florian Lindner, Alexander Rusch, Dmytro Sashko, David Schneider, **Amin Totounferoush,** Dominik Volland, Peter Vollmer and Oguz Ziya Koseomur. "preCICE v2: A Sustainable and User-Friendly Coupling Library." (2021).

9-**Amin Totounferoush,** Axel Schumacher, and Miriam Mehl. "Partitioned Machine Learning of Fluid-Structure Interaction", arXiv preprint arXiv:2105.06785 (2021).

Contents

Abstract iii

Kurzzusammenfassung v

Acknowledgment vii

Publication List ix

1 Introduction 1
- 1.1 Equation Coupling . . . 2
- 1.2 Data Communication . . . 7
- 1.3 Data Mapping . . . 9
- 1.4 Load Balancing . . . 10
- 1.5 Thesis Structure and Contributions . . . 11

2 Efficient and Scalable Communication Initialization with preCICE 13
- 2.1 Introduction to the Coupling Library preCICE . . . 14
- 2.2 Efficient Initialization Using Bounding Boxes . . . 20
  - 2.2.1 Two-level Initialization Scheme . . . 20
  - 2.2.2 Bounding Boxes for Parallel Filtering of Shared Mesh Vertices 21
- 2.3 Performance Results . . . 25
  - 2.3.1 Test Case Description . . . 25
  - 2.3.2 Performance Analysis . . . 27
- 2.4 Summary . . . 37

3 Data-Driven Performance Modeling and Load Balancing 39
- 3.1 Introduction to Load Balancing in Multi-Physics Simulations . . . 40
- 3.2 Performance Modeling . . . 41
  - 3.2.1 PMNF Regression . . . 41
  - 3.2.2 EPMNF Regression . . . 43
  - 3.2.3 Neural Networks . . . 44
- 3.3 Computational Resources Distribution . . . 49
- 3.4 Multi-Step Inter-Solver Load Balancing . . . 52
- 3.5 Load balancing with APES . . . 53

3.6 Performance Results 54
3.6.1 Hardware Description 55
3.6.2 Gaussian Pressure Pulse Test Case 56
3.6.3 Airfoil Test Case 63
3.7 Summary 69

4 Machine Learning for Convergence Acceleration of Coupled Problems 71
4.1 Introduction to the Application of Machine Learning in Simulation Science 72
4.2 Data-Integrated Fluid-Structure Interaction Simulation with Deep Neural Networks 73
4.2.1 On-the-Fly Training to Avoid Model-Data Inconsistency . . . 76
4.2.2 Neural Network Architecture Design and Hyper Parameters' Tuning 76
4.3 Parallel Training Based on Training Data Decomposition 78
4.4 Performance Analysis 80
4.4.1 Data-Integrated FSI Simulation 81
4.4.2 Parallelization of Neural Networks Training for Machine Learning of Partial Differential Equations 90
4.5 Summary 95

5 A Highly Scalable Solver with GPU Acceleration for Partitioned Solution of Fluid-Structure Interaction Problems 97
5.1 Introduction 98
5.2 Governing Equations and Numerical Methods 99
5.2.1 Fluid Solver 99
5.2.2 Structural Solver 102
5.2.3 FSI coupling 103
5.3 Single-Physics Solver Parallelization and GPU Acceleration 105
5.3.1 Single-Physics Solver Parallelization 105
5.3.2 GPU Acceleration of the Fluid Solver 106
5.4 Inter-Solver Parallelization 108
5.4.1 Inter-Solver Point-to-Point Communication 108
5.4.2 Inter-Solver Parallelization in the Presence of GPU Co-Processors 109
5.4.3 Inter-Solver Load Balancing 111
5.5 Performance and Scalability Analysis 112
5.5.1 Test Case Description 112
5.5.2 Hardware Architecture and Numerical Library Description . 114

5.5.3 Strong Scalability on a CPU-Only Machine 115
5.5.4 Strong Scalability on a Hybrid CPU-GPU Machine 118
5.6 Summary 123

6 Summary and Conclusion **125**

Declaration of Authorship **141**

1 Introduction

Multi-physics applications are of increasing interest as computational science communities strive to address broad questions about complex physical and engineered systems characterized by multiple interacting physical processes that have traditionally been considered separately [1]. However, these problems require solutions that span a multitude of physical phenomena, which often can only be solved using simulation techniques that cross several disciplines [2]. This includes a wide range of applications, such as fluid-structure interaction [3, 4, 5, 6, 7], fluid acoustics coupling [8, 9, 10, 11] and conjugate heat transfer [12, 13, 14].

Numerical methods to solve multi-physics problems can be divided into two main categories, monolithic and partitioned. In a monolithic approach, the equations from all occurring physics are discretized and solved as a single large system, inherently accounting for their mutual interaction. Examples of this approach can be found in [15, 16, 17]. The monolithic method provides the opportunity to develop highly tailored and efficient solutions for specific problems. An important disadvantage of this approach is the requirement of developing a new solver and implementing software for each new problem. This significantly increases the software development cost.

In a partitioned approach, on the other hand, the problem is divided into smaller subdomains according to the governing physics, hence, separate solvers are used for each sub-problems. In this setting, a coupling technique must be adopted to account for the interaction of the domains. One of the big advantages of the partitioned approach is the possibility to use the most adapted and well-validated numerical methods for each sub-problem. Moreover, it allows using previously developed and computationally optimized single-physics solver codes, thus saving excessive software development effort [18, 19, 20]. The efficiency of this approach has been shown in many publications [20, 21, 7].

The modern scientific and engineering multi-physics problems that we address are often very complex and require a huge computational effort [7]. Therefore, massively parallel computers must be exploited efficiently for running these simulations. In a partitioned approach, the parallel efficiency of the single-physics solvers plays an important role in the overall performance of the coupled simulations. In recent years, numerous efficient codes have been developed for many single-physics problems,

particularly fluid and structure systems. This includes both open-source solvers such as OpenFoam [22], Calculix [23] and FEniCSx [24] as well as commercial and in-house software codes such as ANSYS Fluent [25] and Termofluids [26, 27].

Nevertheless, using efficient single-physics solvers does not guarantee a scalable coupled simulation. Multi-code coupling introduces several new challenges that must be properly addressed for an efficient simulation. These challenges include, but are not limited to (i) efficient data communication between solvers, (ii) accurately mapping the interface data when solvers use non-matching meshes, (iii) equation coupling at the common interface to ensure the solution is converged, and (iv) load balancing between coupled solvers to maximize the parallel efficiency and minimize idle times. In the following, I briefly introduce these challenges and solutions proposed in the literature so far.

1.1 Equation Coupling

To fulfill the physical equilibrium at the common interface and achieve a converged solution (in each implicit time step or for the stationary solution), the involved solvers perform a kind of fixed-point iteration in a partitioned simulation. To better illustrate this, the coupling in a partitioned solution is explained through an example. Assume we have two solvers coupled in a partitioned simulation, say $S1$ and $S2$. These two can be for example a fluid and a solid solver in a fluid-structure interaction (FSI) problem. The solvers can be modeled as two operators $S_1 : X_1 \rightarrow X_2$ and $S_2 : X_2 \rightarrow X_1$, each of them taking the output of the other one at the coupling interface as input. In an FSI problem, the fluid solver S_1 takes the displacement at the common interface $x_1 \in X_1$ as the input, and calculates the corresponding pressure distribution at the boundary $x_2 \in X_2$. In return, the solid solver S_2 takes x_2 as an input to calculate the displacement.

The solvers can be called either in a parallel or a serial manner. In the former, both solvers are called simultaneously, while in the latter they run one after the other. In the case of a serial coupling scheme, S_1 and S_2 are applied consecutively. This process can be presented as the following fixed point iteration:

$$x_1^{\text{new}} = \text{ACC} \circ S_2 \circ S_1(x_1). \tag{1.1}$$

On the other hand, in a parallel coupling scheme, we apply S_1 and S_2 simultaneously to a set of input vectors X_1 and X_2:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^{\text{new}} = \text{ACC} \circ \begin{pmatrix} S_2(x_2) \\ S_1(x_1) \end{pmatrix}. \tag{1.2}$$

ACC denotes a convergence acceleration method. For a simple fixed-point iteration, ACC is the identity. More sophisticated approaches are under-relaxation or quasi-Newton which will be explained later in this section.

In addition to the categorization into the parallel and sequential coupling, coupling schemes can be categorized into explicit and implicit methods. In the former, solvers are called for a fixed number of times (iterations of the fixed point equation) during each time step disregarding the convergence of the solution at the boundary. This can lead to numerical instabilities and inaccurate solutions if there is a strong physical coupling between domains. Implicit schemes, on the other hand, iterate until convergence, meaning that further iterations do not change the solution. These schemes prevent numerical instabilities and produce more accurate solutions. However, due to the higher number of iterations, these schemes are computationally more expensive.

Combining the mentioned categorizations results in four different coupling schemes: (i) explicit-serial, (ii) explicit-parallel, (iii) implicit-serial, and (iv) implicit-parallel. In the following, each one is explained briefly.

In an **explicit-serial** scheme the first solver S_1 uses the boundary values of the previous time step $x_1^{(t-1)}$ to solve the domain equations and output new coupling values $x_2^{(t)}$. When the first solver has finished, the second solver S_2 starts the solution with the new coupling values $x_2^{(t)}$. This process is repeated until the end of the simulation $t_{\max}$ is reached. Algorithm 1 expresses this scenario as a pseudo-code.

Algorithm 1 Pseudo-code: explicit-serial coupling scheme for the partitioned simulation.

```
initialize x_1^(0)
for t = 1, ..., t_max do
    x_2^(t) ← S_1(x_1^(t-1))
    x_1^(t) ← S_2(x_2^(t))
end for
```

In an **explicit-parallel** scheme, the coupled solvers ($S1$ and $S2$) run simultaneously using the values of the previous time step ($x_1^{(t-1)}$, $x_2^{(t-1)}$), i.e., lines 3 and 4 in Alg. 2

are executed at the same time.

Algorithm 2 Pseudo-code: explicit-parallel coupling scheme for the partitioned simulation.

initialize $x_1^{(0)}, x_2^{(0)}$
for $t = 1, \ldots, t_{\max}$ **do**
    $x_2^{(t)} \leftarrow S_1(x_1^{(t-1)})$
    $x_1^{(t)} \leftarrow S_2(x_2^{(t-1)})$
end for

The **implicit-serial** scheme is similar to the explicit-serial in the sense that coupled solvers are called one after the other. The difference, however, is that for each time step, the solution is repeated until a convergence criterion is met. This scenario is depicted in Alg. 3.

Algorithm 3 Pseudo-code: implicit-serial coupling scheme for the partitioned simulation.

initialize $x_1^{(0),0}$
for $t = 1, \ldots, t_{\max}$ **do**
    $j \leftarrow 0$
    while not converged **do**
        $x_2^{(t),j+1} \leftarrow S_1(x_1^{(t),j})$
        $x_1^{(t),j+1} \leftarrow S_2(x_2^{(t),j+1})$
        $j \leftarrow j + 1$
    end while
    $x_1^{(t+1),0} \leftarrow x_1^{(t),j}$ ▷ assign starting value for next iteration
end for

Finally, in an **implicit-parallel** coupling scheme, the solvers are called simultaneously using the values of the previous iteration. This process is repeated within each time step until the solution converges. The algorithm is shown on Alg. 4.

In case of using an implicit coupling scheme, either serial or parallel, the fixed-point equation are commonly accelerated with Aitken's relaxation (e.g. [18, 28, 29]) or Newton-based methods (e.g. [30, 31, 32]) which is not included in Alg. 1–4. Fixed-point iterations with Aitken's relaxation are seen to be efficient and robust in some serial coupling solution cases (see e.g. [28]). However, they often lead to robust, but still slow convergence, and their performance decreases in a parallel (simultaneous) coupling solution. On the other hand, parallel execution together with a quasi-Newton method is very efficient and robust [33, 21]. Therefore, in the current work, the quasi-Newton method is used to accelerate implicitly coupled solvers.

Algorithm 4 Pseudo-code: implicit-parallel coupling scheme for the partitioned simulation.

initialize $x_1^{(0),0}, x_2^{(0),0}$
for $t = 1, \ldots, t_{\max}$ **do**
  $j \leftarrow 0$
  while not converged **do**
    $x_2^{(t),j+1} \leftarrow S_1(x_1^{(t),j})$
    $x_1^{(t),j+1} \leftarrow S_2(x_2^{(t),j})$
    $j \leftarrow j + 1$
  end while
  $x_1^{(t+1),0} \leftarrow x_1^{(t),j}$ ▷ assign starting values for next iteration
  $x_2^{(t+1),0} \leftarrow x_2^{(t),j}$
end for

In the following, the quasi-Newton acceleration method is explained in a parallel coupling setting. Our short introduction follows the description in [3], a more detailed introduction of the method can be found in [32, 34, 35]. One iteration of the parallel execution of the fixed-point equation problem can be written in matrix-like notation as

$$\begin{pmatrix} \widetilde{\mathbf{x}}_1^{k+1} \\ \widetilde{\mathbf{x}}_2^{k+1} \end{pmatrix} = \begin{pmatrix} 0 & S_2 \\ S_1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x}_1^k \\ \mathbf{x}_2^k \end{pmatrix}, \tag{1.3}$$

where k indicates the iteration count in the current time step $t^{n-1} \rightarrow t^n$, and the tilde sign in $\widetilde{\mathbf{x}}_1^{k+1}$ and $\widetilde{\mathbf{x}}_2^{k+1}$ means these new values are not converged yet and must be modified in a subsequent quasi-Newton step. The vector form of the underlying fixed-point equation (Eq. (1.3)) can be shortly written as

$$\mathbf{x} = \mathcal{H}(\mathbf{x}), \tag{1.4}$$

where $\mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix}$ and $\mathcal{H} = \begin{pmatrix} 0 & S_2 \\ S_1 & 0 \end{pmatrix}$. To solve Eq. (1.4), the fixed-point iteration (Eq. (1.3)) can be accelerated by a subsequent Newton step

$$\mathbf{x}^{k+1} = \mathcal{H}(\mathbf{x}^k) - \mathcal{J}^{-1}\mathbf{R}(\mathbf{x}^k) \tag{1.5}$$

where k denotes the iteration count, $\mathbf{R}$ is the residual function $\mathbf{R}(\mathbf{x}^k) = \mathcal{H}(\mathbf{x}^k) - \mathbf{x}^k$, and $\mathcal{J}^{-1}$ is the inverse Jacobian of $\widetilde{\mathbf{R}}(\mathbf{x}^k) = \mathbf{x}^k - \mathcal{H}^{-1}(\mathbf{x}^k)$. Since the calculation of the

inverse Jacobian is not feasible, it is approximated based on the secant equation

$$\hat{\mathcal{J}}_k^{-1}\mathcal{V}_k = \mathcal{W}_k \tag{1.6}$$

in which the hat sign indicates the approximation and $\mathcal{W}_k$ and $\mathcal{V}_k$ are two matrices which include increments of $\widetilde{\mathbf{x}} = \mathcal{H}(\mathbf{x})$ and the residual $\mathbf{R}$ collected from the previous iterations

$$\mathcal{W}_k = [\Delta\widetilde{\mathbf{x}}^k, \Delta\widetilde{\mathbf{x}}^{k-1}, ..., \Delta\widetilde{\mathbf{x}}^1] \tag{1.7}$$

$$\mathcal{V}_k = [\Delta\mathbf{R}^k, \Delta\mathbf{R}^{k-1}, ..., \Delta\mathbf{R}^1] \tag{1.8}$$

with $\Delta\widetilde{\mathbf{x}}^k = \widetilde{\mathbf{x}}^i - \widetilde{\mathbf{x}}^{i-1}$ and $\Delta\mathbf{R}^k = \mathbf{R}^i - \mathbf{R}^{i-1}$. There are various options to solve the underdetermined system 1.6. The classical interface quasi-Newton (IQN) approach uses minimization of the Frobenius norm as presented in [33, 36, 35] to close the system:

$$\left\|\hat{\mathcal{J}}_k^{-1}\right\|_F \rightarrow min., \tag{1.9}$$

which results in the following Jacobian estimation:

$$\hat{\mathcal{J}}_k^{-1} = \mathcal{W}_k(\mathcal{V}_k^T\mathcal{V}_k)^{-1}\mathcal{V}_k^T. \tag{1.10}$$

Alternatively, we can use a multi-vector quasi-Newton method (IMVJ) as proposed in [37, 32]. In this approach, instead of minimizing the Frobenius norm of the Jacobian, the distance between the current and the previous time step's Jacobian ($\hat{\mathcal{J}}_k^{-1}$ and $\hat{\mathcal{J}}^{-1,(n)}$, respectively) is minimized ($||\hat{\mathcal{J}}_k^{-1} - \hat{\mathcal{J}}^{-1,(n)}|| \rightarrow min.$). This results in the following approximation for the Jacobian inverse:

$$\hat{\mathcal{J}}_k^{-1} = \hat{\mathcal{J}}^{-1,(n)} + (\mathcal{W}_k - \hat{\mathcal{J}}^{-1,(n)}\mathcal{V}_k)(\mathcal{V}_k^T\mathcal{V}_k)^{-1}\mathcal{V}_k^T, \tag{1.11}$$

where k indicates the iteration number at the current time step, while n refers to the previous time step t^n. It should be noted that we cannot always use all available data from previous iterations, since this might induce linear dependencies in the columns of V, which leads to an ill-posed problem. Therefore, linearly dependent data must be removed by using a proper filter [38, 32, 37].

The estimated Jacobian is used to find the increment of $\mathbf{x}$, according to Eq. (1.5). This process must be repeated until the convergence criterion is met and we can move to

the next time step. The convergence criterion is defined as

$$\frac{||\mathbf{R}^k||}{||\mathbf{R}^0||} < \epsilon_{FSI} \tag{1.12}$$

with a prescribed small value for ϵ_{FSI}.

In the first iteration of the first time step, there is no previous data available to use the quasi-Newton method. Therefore, the fixed-point equation can be accelerated with an under-relaxation. In addition, at the beginning of each time step, a second-order extrapolation is used to create a better initial guess which helps to decrease the number of required iterations.

Note that the serial scheme usually requires fewer iterations, however, it always idles one solver. On the other hand, the parallel scheme requires a slightly higher number of iterations, but it does not idle any solver. The parallel scheme has been shown to be more efficient for highly parallel simulations. In addition, for coupling three or more solvers, the parallel coupling scheme is numerically more efficient since the acceleration is applied to all coupling variables. Furthermore, it is technically more straightforward as it treats all coupling connections in an equal manner.

1.2 Data Communication

Inter-solver data communication is an important component of a partitioned simulation. Efficient handling of this communication can significantly reduce the coupling cost. In a partitioned simulation, coupled single-physics solvers need to exchange coupling data, which is usually handled by an external coupling library. These libraries usually use either message passing interface (MPI) or lower-level TCP/IP sockets for inter-solver communication. In general, there are two main models to establish inter-solver communication: (i) the centralized intercommunication model (CICM) and (ii) the distributed intercommunication model (DICM). While the former (Fig. 1.1) employs a central entity as a server to handle communications of other entities (clients), the latter (Fig. 1.2) allows entities to directly exchange data between each other [39]. Both models have been used in available coupling libraries. For example, MpCCI [40], OpenPALM [41] and EMPIRE [42] use a CICM model, while preCICE [34] and OASIS3-MCT [43] employ a DICM model. A DICM model has many advantages over a CICM model:

1. The direct message communication is faster since it does not travel over an extra entity.
2. It has a higher potential for parallelization since the message exchange is performed point-to-point.
3. A DICM model avoids any memory issue that a CICM model can suffer from the accumulated messages from clients.
4. The load can be better balanced since there is no central entity that can be overloaded with messages (In the cases that the central entity involves also in computations, its load can be much higher than that of other entities).

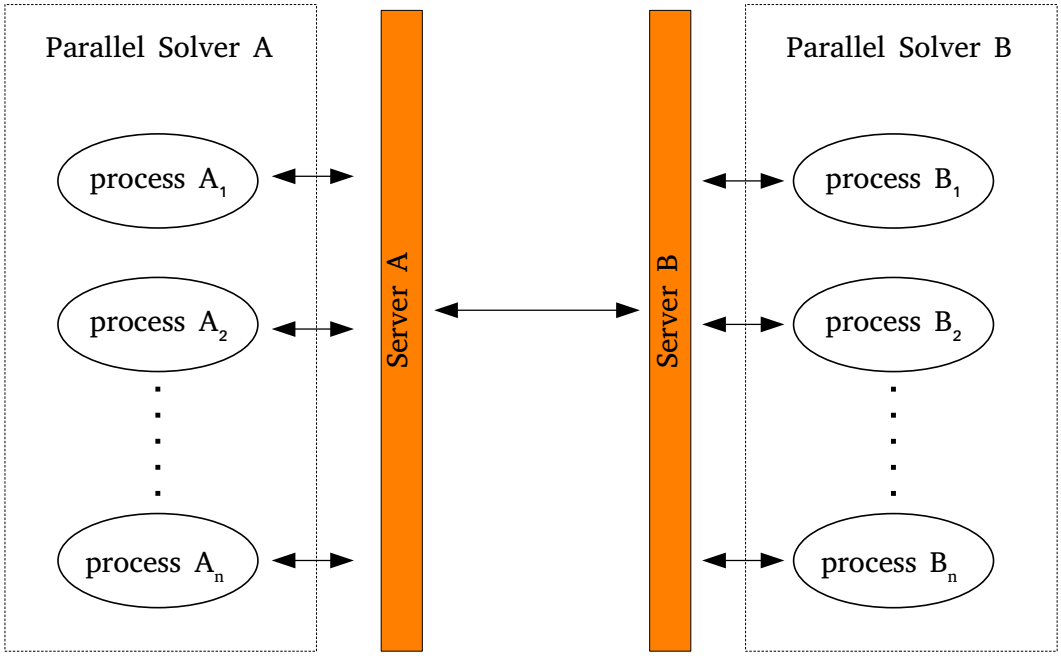


Figure 1.1: Inter-solver data communication models: centralized intercommunication model (CICM).

Inter-solver communication can be also categorized based on the communication mode, which can be blocking or non-blocking. In a blocking mode, the operation does not return until it is completed. For example, when calling `MPI_Recv` in an MPI communication, the operation is complete only when the message has been received. On the other hand, a non-blocking operation returns immediately, letting other operations continue [44]. The DICM model can be used efficiently only in conjunction with non-blocking operations. Using blocking operations sequentializes the communication, since these operations can only be performed one after the other, i.e., one blocking operation must be finished before starting to call the next one.

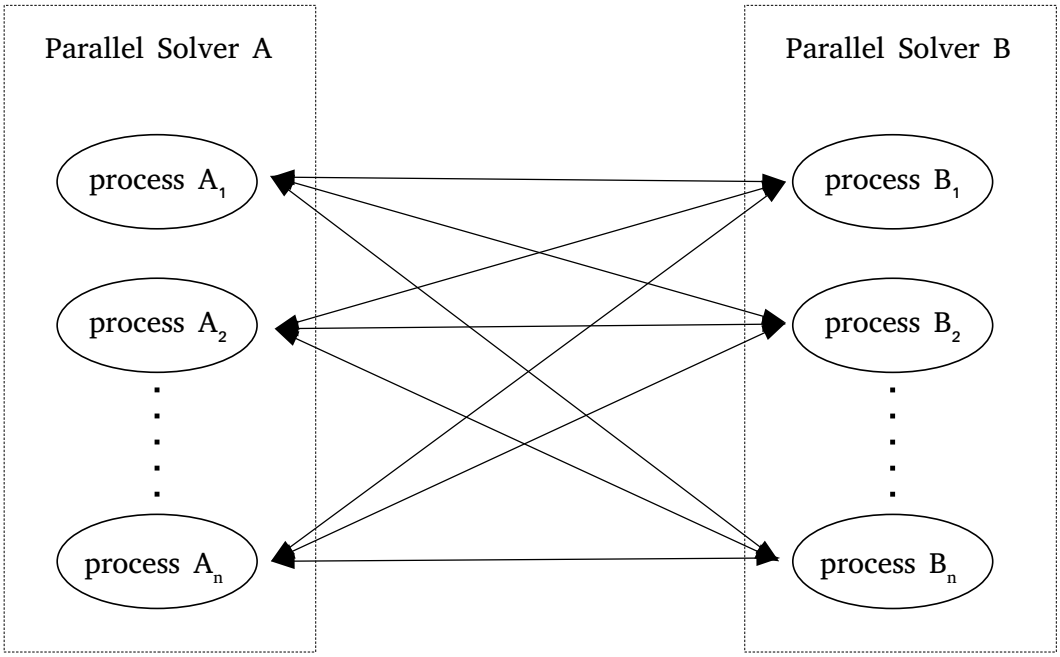


Figure 1.2: Inter-solver data communication models: distributed intercommunication model (DICM).

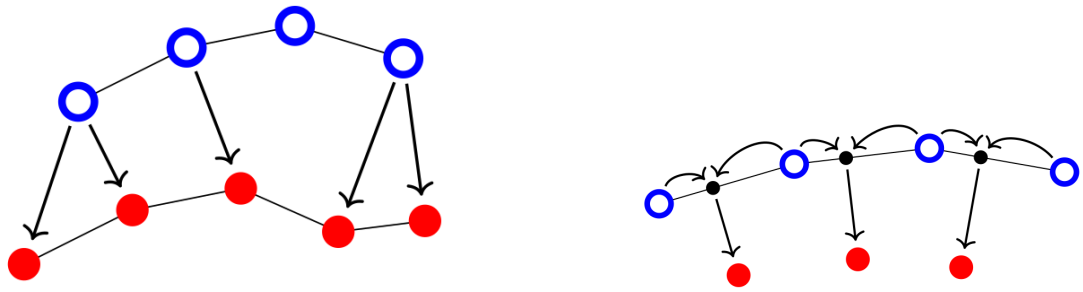
1.3 Data Mapping

In a partitioned coupled simulation, it is required that common interface values are transmitted between domains. For instance, in a fluid-structure interaction problem, once the displacement of the structure is computed, it must be imposed on the fluid domain. In general, it is not feasible to generate matching interface meshes for coupled problems. This is because solvers have different requirements for their computational mesh. They may need different interpolation scheme order within a single element. In addition, depending on the geometry and the governing physics, different mesh resolutions can be required. Therefore, when meshes are non-matching, an interpolation/projection step has to be performed to enable an accurate information transfer between domains [45, 46].

In literature, various methods have been proposed such as nearest neighbor interpolation [46], projection methods [47, 48] and interpolation methods based on radial basis functions (RBF) [49, 50].

Nearest neighbor mapping (Fig. 1.3a) is a simple mapping method in which data are copied from nearest neighbor points in the partner mesh (for example from mesh A to mesh B). A search algorithm is needed to find the closest point in mesh A (x_A) to a given point in mesh B (x_B). The variable in x_A is then transferred to x_B.

Projection based mapping (Fig. 1.3b) is based on the projection of data points of mesh B to mesh A. Instead of taking the values from the closest point x_A, the point x_B is orthogonally projected on mesh A and the reconstructed value of the quantity of interest in that point is taken.
RBF based mapping uses radial basis functions centered at the source data points as basis functions for a global interpolation function [46].



(a) Nearest neighbor mapping.

(b) Nearest projection mapping.

Figure 1.3: Schematic view of consistent projection-based mappings in a two-dimensional case. Arrows show the data transfer direction. This illustration has been taken from [51].

All these mapping methods can be used in both conservative and consistent forms. The conservative form preserves integral values of the data, whereas the consistent form reproduces constant functions on the mapping surface exactly. Consistent mapping is used for data such as displacements in which the point-value is of importance, whereas conservative mapping is suitable for data such as forces for which the integral value must be preserved.

1.4 Load Balancing

Running a partitioned simulation on a massively parallel machine requires that the coupling data are communicated between different nodes. In a black-box partitioned framework, one `mpiexec` command per solver is called which pins an independent set of `MPI` processes to each solver. Therefore, processes can not be shared between solvers and, thus, must be distributed among them. This is schematically shown in Fig. 1.4.

Depending on the coupling strategy, parallel or serial, the distribution of computational resources must be optimized to maximize the parallel efficiency. An imbal-

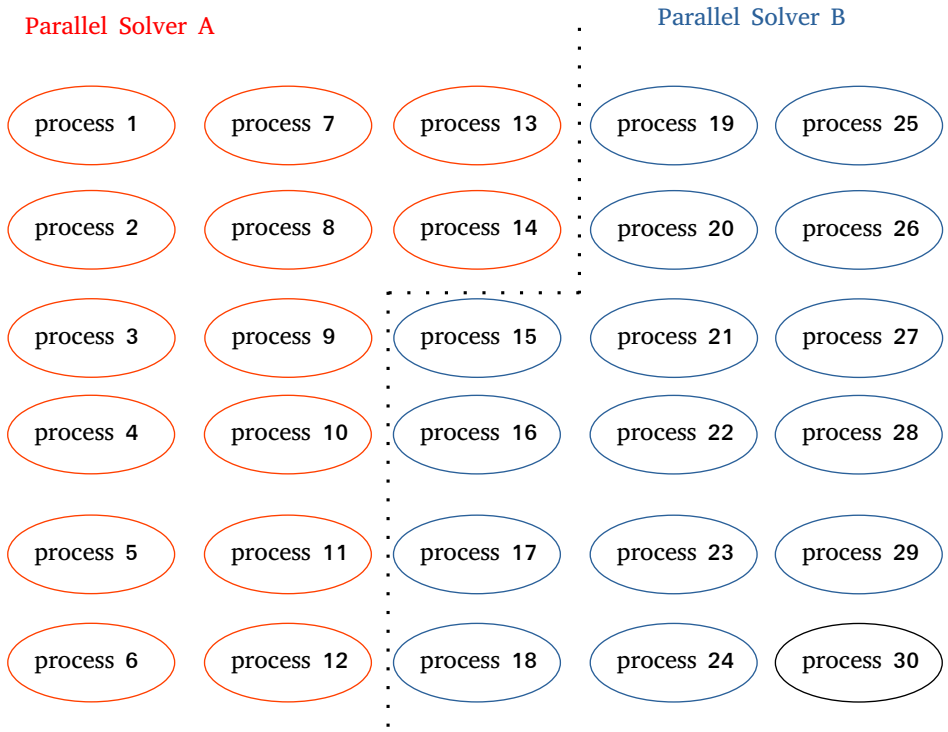


Figure 1.4: Inter-solver load balancing: distribution of computational resources between two solvers A and B.

anced distribution can result in a significant performance drop. Note that this is a different issue than load balancing within processes of a single solver. Each process's load per unit (e.g., per mesh element) must be measured to evenly distribute the computation load among processes. Even though load balancing within a single solver has been extensively studied, see for example [52, 53], to the best of my knowledge, the inter-solver load balancing for partitioned coupled simulations is largely neglected in the literature. The main issue here is that the coupled solvers use different degrees of hardware optimizations, different numerical methods, unpredictable numbers of iterations per time step and equation systems, etc., which make using analytical or heuristic weights per discretization point for balancing the load across solvers impossible.

1.5 Thesis Structure and Contributions

This thesis presents methods to improve the computational efficiency of partitioned multi-physics simulations by addressing the remaining issues in data communication, load balancing, and equation coupling. Inter-solver data communication has

been properly addressed in the literature. Implementing point-to-point communication between partner ranks of coupled solvers in the preCICE coupling library has removed the performance and memory bottleneck of central communication. The implementation is very efficient on a large number of CPU cores [54]. However, the initialization of communication is still performed sequentially. To effectively address this issue, chapter 2 introduces a two-level efficient initialization scheme for partitioned multi-physics simulations.

As explained earlier, the inter-solver load balancing is an open issue for partitioned simulations. Chapter 3 addresses this challenge by incorporating a data-driven performance modeling and optimizing the distribution of computational resources among coupled single-physics solvers. For performance modeling, two different methods are used. For problems with small to medium problem sizes, a single variable regression method is adapted. The overhead of performance modeling for problems with large mesh sizes using this regression is considerable, even prohibitive. Therefore, multi-variable modeling methods using regression and neural networks are introduced. The neural networks alternative can generate accurate but cheaper performance models for the problems with a large computational mesh. Based on these performance models and according to the coupling scheme (parallel or serial), an optimization problem is derived and solved to calculate the optimal load balancing between coupled single-physics solvers.

For equation coupling in strongly coupled problems, Quasi-Newton methods have shown promising performance. However, there exist cases where these advanced methods still require a high number of iterations to produce converged solutions. To address these cases, Chapter 4 introduces a machine learning-based idea to further accelerate the solution convergence. In this method, we use a combination of convolutional neural networks and recurrent layers to estimate the coupled problem's solution. We show that feeding this estimation to the numerical solvers along with an advanced quasi-Newton acceleration method can further improve the convergence speed.

To demonstrate the high efficiency of the partitioned method and improvements made by the methods introduced in this thesis, a highly parallel FSI framework with GPU-acceleration capability is introduced in chapter 5. The framework couples highly parallel solvers using preCICE library and incorporates the load balancing scheme introduced in chapter 3 to investigate complex problems in the field of hemodynamics simulation. Finally, chapter 6 summarizes and concludes the thesis.

2 Efficient and Scalable Communication Initialization with preCICE

preCICE is an open-source library that provides the required functionality to couple independent parallel solvers to establish a partitioned multi-physics multi-code simulation environment. For data communication between the respective executables at run time, preCICE implements a point-to-point scheme, where ranks of coupled solvers directly exchange data avoiding a central communication instance. This drastically reduces the cost of data communication during the simulation. To initialize the coupling, mesh partitions of the respective solvers need to be compared to determine the point-to-point communication channels between the processes of both codes. In the case of performing the mesh communication in a gather-scatter manner, this initialization effort can become a limiting factor, if we either reach memory limits or if we have to re-initialize communication relations in every time step. In the current chapter, it is explained how the sequential gather-scatter comparison of mesh partitions is replaced by a two-level approach that first compares bounding boxes around mesh partitions in a sequential manner, uses the results to establish pairwise communication between processes of the two solvers, and finally compares mesh partitions between connected processes in parallel. It is shown that the two-level initialization method is five times faster than the old one-level scheme on 24,576 CPU cores using a mesh with 628,898 vertices. In addition, the two-level scheme can handle much larger computational meshes, since the central mesh communication of the one-level scheme is replaced by a fully point-to-point mesh communication scheme [55]. The remainder of this chapter is organized as follows. Section 2.1 introduces the communication schemes in preCICE with a focus on the initialization phase. The new initialization method is explained in Sec. 2.2. Scalability and efficiency measurements and analysis are presented in Sec. 2.3, while Sec. 2.4 summarizes and concludes the chapter.

2.1 Introduction to the Coupling Library preCICE

The coupling library preCICE has been developed to couple different solvers in multi-physics simulations and to (1) handle the communication between different solvers, (2) provide data mapping for the coupling data, and (3) offer iterative equation coupling schemes for surface coupled problems in a modular way. The major components and the library concept of preCICE are shown in Fig. 2.1.

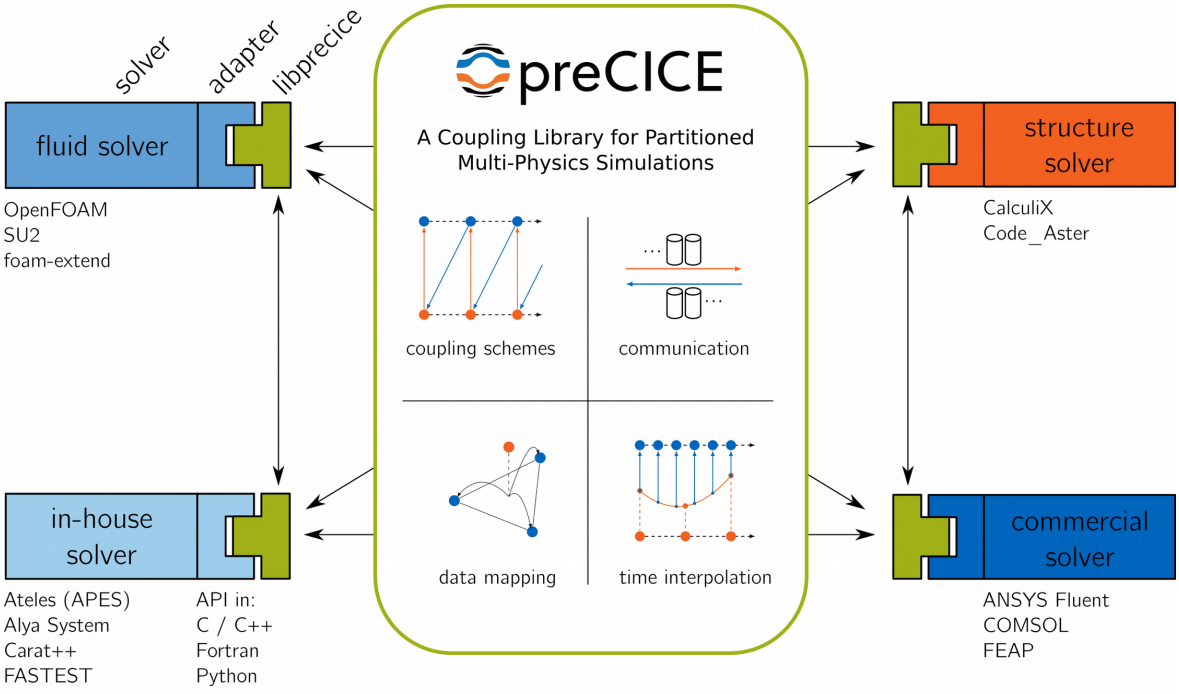
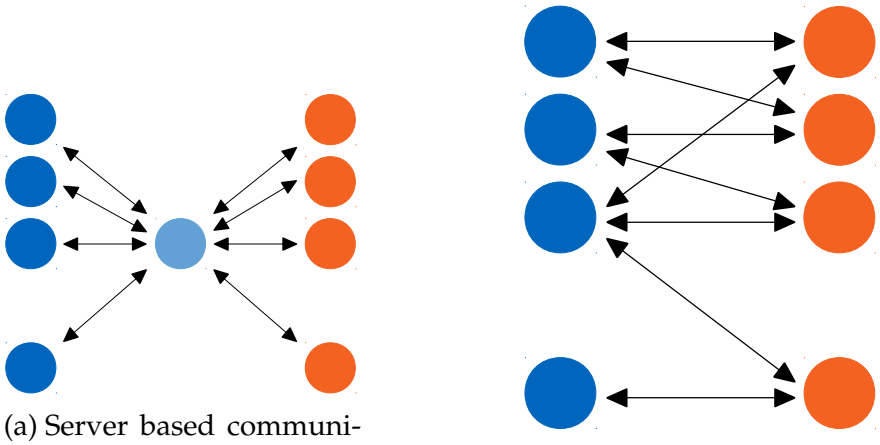


Figure 2.1: Main components and the library concept of the preCICE coupling library. preCICE provides data communication, equation coupling and data mapping between non-matching meshes for partitioned coupled multi-physics simulations. This illustration has been taken from [34].

Communication: For multi-physics problems, different solvers need to communicate with each other in order to exchange coupling data and get the required information at the common boundaries. preCICE establishes the necessary communication channels between different ranks of the involved solvers. The inter-code communication is based on either MPI ports (MPI-2.0) or lower-level TCP/IP sockets. The latter, even though slower, is particularly important on HPC systems as many MPI implementations do not support MPI ports functionality. preCICE does not use any central communication unit (Fig. 2.2a) and run-time communication is fully parallel point-to-point (Fig. 2.2b), i.e., each rank directly communicates with the connected



(a) Server based communication: communication between partner ranks is peformed via a server rank.

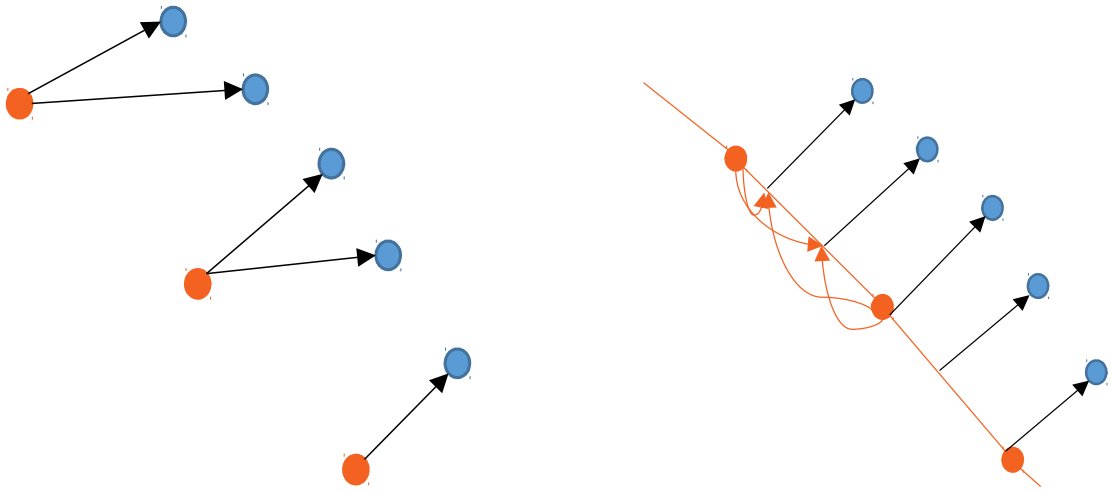
(b) Point-to-point communication: partner ranks directly communicate with each other.

Figure 2.2: General concepts for the communications between ranks of coupled solvers can be performed either:(a) via a server rank or (b) directly between partner ranks. preCICE uses the latter concept.

ranks of the other solver [54].

Data Mapping: Due to non-matching meshes used by various solvers, the communicated coupling data of the sending solver must be mapped to the right points in the mesh points of the receiving solver. preCICE offers various mapping methods which can be selected according to the physical constraints of the simulation. In particular, all mapping methods are available both as conservative and as consistent variants. The conservative form preserves integral values of the data, whereas the consistent form reproduces constant functions on the mapping surface exactly. The consistent mapping is used for data such as displacements in which the point-value is of importance, whereas the conservative mapping is suitable for data such as forces for which the integral value must be preserved. We briefly explain the different mapping methods implemented in preCICE [34]. **Nearest Neighbor (NN)** mapping (Fig. 2.3a) is the simplest method that needs only vertex positions and copies the data to the closest data point of the partner mesh. **Nearest Projection (NP)** mapping (Fig. 2.3b) is based on the projection of data points of the receiving solver's mesh to the sending solver's mesh and a second-order interpolation. This mapping requires topology information of the source mesh, i.e., a surface triangulation. **Radial Basis Function (RBF)** mapping uses radial basis functions centered at the source data points for a proper data mapping. This method needs no topological information.

Equation Coupling: Depending on the physical coupling strength between coupled



(a) Nearest neighbor mapping. (b) Nearest projection mapping.

Figure 2.3: Schematic view of consistent projection-based mappings in a two-dimensional case. Arrows show the data transfer direction. This illustration has been taken from [54].

fields, executing a certain time step might require iterating for a fixed small number of iterations or for a higher number of iterations to converge to the implicitly coupled solution. These two scenarios are known as explicit and implicit coupling, which are both supported by preCICE. It must be pointed out, that these two alternatives can be used in both serial and parallel schemes. In serial schemes, one solver waits until the other one finishes one iteration, while in the parallel scheme, both solvers run simultaneously and exchange the coupling data at the end of each iteration/time step. Various convergence acceleration methods such as Aitken and quasi-Newton are implemented in preCICE. More details can be found in [54, 32].

As already mentioned, preCICE uses a peer-to-peer communication concept to couple MPI-parallel codes and avoids using any central communication instance for data exchange. In addition, coupling numerics (such as radial-basis function interpolation or quasi-Newton acceleration) are computed directly within the library on the processes of the coupled codes [56]. This makes the computational cost of the actual coupling per time step negligible compared to the typical run time of the coupled codes themselves, even for very large cases on ten thousand processes [54].

In the following, we focus on the initialization. The purpose of the initialization is to determine partner ranks between coupled solvers and identify the exact list of data that must be communicated between partner ranks during the run time. preCICE previously used a one-level initialization scheme implementation by Uekermann [54]. The one-level approach and the methods implemented in similar cou-

pling libraries have been explained in [57]. This explanation is extended in the remainder of this section for a better understanding. To better explain the one-level approach, we provide an example which is schematically shown in Fig. 2.4. In this implementation, the interface mesh partitions of one solver (B in this example) are gathered in the master rank (step I) and communicated to the ranks of the other solver (A in this example) via a master-to-master communication (step II) followed by a broadcast (step III). Each rank of A can compare its mesh partition to the received mesh and identify the list of partner ranks (in solver B) and the data that must be communicated during the run time according to the selected mapping scheme (step IV). Therefore, the chosen data mapping scheme and the mesh geometry strongly impact this filtering. Finally, this information is gathered in the master rank of A (step V) and communicated to the ranks of solver B via a master-to-master communication followed by a broadcast, so that ranks of solver B also have access to this information.

For test cases with a relatively small computational mesh and those exploiting up to a few hundred processes, the performance of the initialization of the coupling is non-critical. Considering this and given the fact that a very high number of time steps is computed after a single initialization, the initial implementation by Uekermann (one-level scheme) can be efficiently used for many coupled simulations. However, the initialization can become a limiting factor, if we either reach memory limits in the gather-scatter algorithm or if we have to re-initialize communication relations due to dynamic changes in mutual data dependencies between coupled codes.

To this end, Lindner [51, 11] already removed several initialization bottlenecks concerning the creation of communication channels on the one hand and neighborhood search between data points of mesh partitions of two solvers on the other hand. I briefly summarize these contributions in the following two paragraphs.

As already mentioned, preCICE offers two communication backends to realize communication between two codes: MPI and Transmission Control Protocol (TCP)/IP. In general, the coupled parallel codes use MPI for their internal communication. If MPI is chosen as the communication backend, inter-communicators between the ranks of the two MPI worlds of the solvers are created through MPI ports to connect the coupled codes. These extra MPI communicators include interface processes from both solvers, in contrast to the solvers' MPI communicator which only includes the processes of the same solver. These extra communicators are needed for data exchange among processes of coupled solvers. Previously, one communicator per process of one solver existed which contains only this process along with all its partner ranks

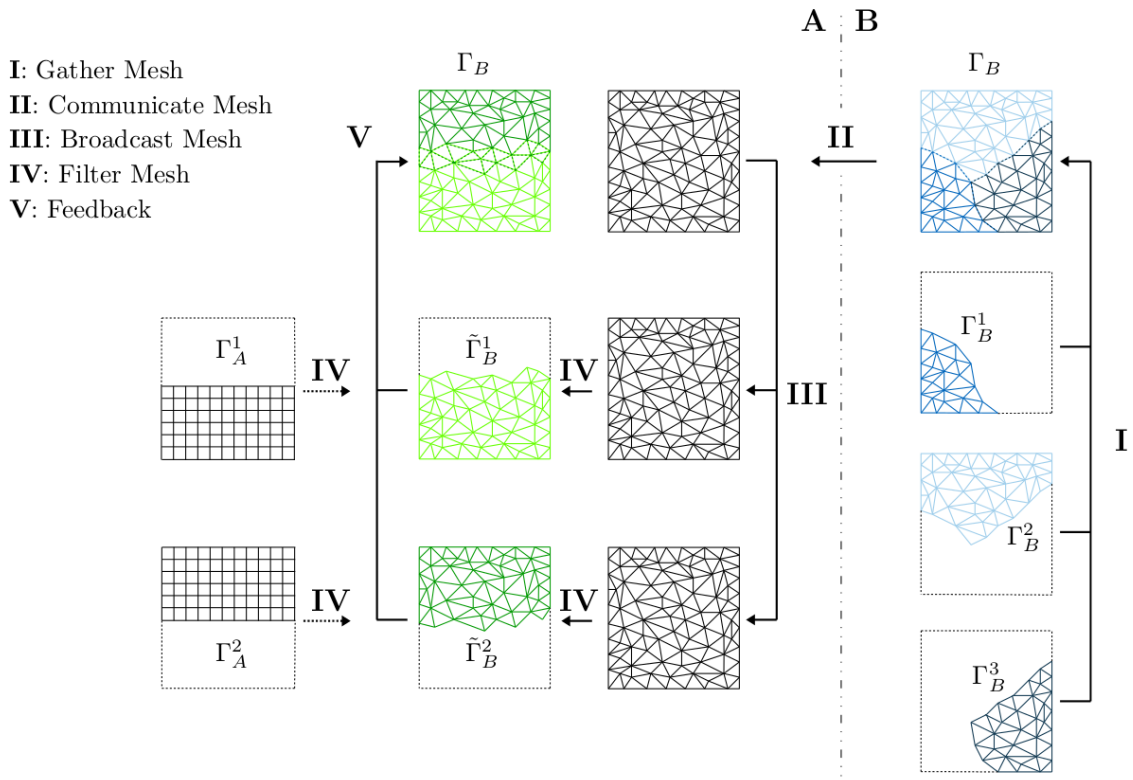


Figure 2.4: Previous one-level initialization scheme of preCICE: (I) Interface mesh partitions of solver B are gathered in the master rank, (II) communicated to the master rank of solver A, and (III) broadcasted to the ranks of A. (IV) Each rank of A compares its mesh partition to the received mesh to identify the list of partner ranks and the list of data that must be communicated during run time. (V) This information is gathered in the master rank of A and communicated to the ranks of B. This illustration has been taken from [54].

of the other solver. This resulted in a very large number of extra communicators and thus increased the total initialization time. Lindner replaced this concept by using a single large communicator including interface ranks of both solvers. This reduced the cost of the creation of the communication channels and also improved the efficiency of the actual data communication on some high-performance computing architectures. To establish TCP/IP-based connections, each pair of connected processes needs to exchange a connection token via the file system. Storing all tokens in a single directory exerts a heavy load on the file system when a large number of communication pairs exist. To reduce the load on the file system, Lindner introduced a hash-based scheme, which distributes the connection files among different directories in an optimal way. This reduces the file system load significantly.

In terms of the mapping itself, the bottleneck after establishing communication channels between the processes is the comparison of mesh partitions of the two solvers to determine the exact data dependencies between mesh entities (elements, vertices, ...) of the two non-matching meshes. For all three data mapping methods that preCICE offers (i.e. nearest neighbor, nearest projection, and RBF), a neighborhood search of mesh vertices must be carried out during initialization. Lindner reduced the cost of this neighborhood search for the nearest-neighbor mapping from $O(n^2)$ to $O(n \log(n))$ by introducing a tree-based search scheme. Similarly, he managed to speed up both the neighborhood search and the initialization of the RBF interpolation system matrix.

This chapter tackles another bottleneck by removing the remaining gather-scatter components of the initialization, which still hinder very large coupled simulations. The one-level gather-scatter approach is replaced by a two-level scheme. On the first level, only bounding boxes around mesh partitions are communicated in a gather-scatter manner to determine preliminary communication channels. On the second level, potential partner ranks directly communicate full mesh data for final filtering and determining the final list of communication channels and mesh dependencies.

Before we explain the new concept in detail, we give an overview of the communication and data mapping initialization strategies of similar coupling software and briefly compare them to ours. The commercial tool MpCCI [58] initializes the coupling on a centralized coupling server [40], which degrades scalability [54]. The initialization process can, however, be repeated in case of re-meshing in one of the coupled codes. DTK [59] creates a third rendezvous decomposition on additional coupling processors to correlate two independently decomposed meshes. To this end, a recursive coordinate bisectioning algorithm is used, which can completely operate on distributed mesh structures [60]. OpenPALM [41] follows a similar bounding-box approach for comparing mesh partitions as the one we propose in this contribution. Afterward, octree-based data structures are used to accelerate the search process within each mesh partition. MUI [61] does not create mesh structures at initialization but computes on-the-fly data mapping in every coupling step. This allows to couple dynamically changing meshes and particle codes. To avoid all-to-all communication, each rank can optionally define additional regions of interest, which are compared during initialization similar to our bounding-box comparison. Finally, CUPyDO [62] follows a similar approach as preCICE previously used [54]: mesh partitions are gathered and scattered by a single processor during initialization.

2.2 Efficient Initialization Using Bounding Boxes

This section explains how bounding boxes can be used for efficient communication initialization. We first explain the procedure in which: (1) the communication channels between partner ranks of coupled solvers are identified and established and (2) the exact list of the data that must be communicated during run time is obtained according to the selected mapping scheme. In the end, we explain a very specific use case of bounding boxes to parallelize filtering of mesh vertices that are shared among several processes of a solver in the RBF mapping implementation in preCICE.

2.2.1 Two-level Initialization Scheme

The two-level scheme is introduced to effectively address the scalability and memory issues of the communication initialization in preCICE, explained in Sec. 2.1. This scheme breaks down the initialization into two levels. The first level identifies and establishes potentially required communication channels, while the second level specifies the actual list of data to be exchanged. Both levels have been initially described in [57]. This section extends this description for a better understanding.

In the first level, each process located at the common interface computes a bounding box around its coupling mesh partition, see Fig. 2.5. The bounding box is defined by the range of x-, y-, and z-coordinates of the respective mesh partition. The bounding boxes of all processes with a non-empty coupling region (coupling processes) are gathered in a master process of one solver (solver B in the example of Fig. 2.5). This corresponds to step I in Fig. 2.5. The set of bounding boxes is communicated to the other solver via master-to-master communication and then broadcast to all coupling processes of the receiving solver (steps II and III in Fig. 2.5). Each process of the receiving solver (solver A) compares its bounding box with the received set to identify relevant partner processes with mesh partitions that potentially interact with the own coupling mesh partition in the given data mapping (step IV). The information about the partner ranks and potentially required channels are gathered in the master rank of A (step V), communicated to the ranks of solver B via a master-to-master communication (step IV) followed by a broadcast to the ranks of solver B (step VII). This level provides the required information regarding the list of connected processes to establish communication channels. preCICE uses this information to establish the communication channels. Some of these connections may be omitted if the list of data to be communicated is found to be empty later. preCICE offers two options

to establish communication channels: (i) based on MPI via MPI ports or (ii) using lower-level TCP/IP sockets. For MPI-based communication, preCICE constitutes a single extra MPI communicator including all coupling processes of both solvers. For the TCP/IP-based communication, a hash-based directory and file naming scheme is implemented to store the connection tokens in an optimally distributed way. This reduces the load on the file system substantially. More details on creating communication channels in preCICE can be found in [51].

For storing and communicating bounding boxes, preCICE uses a C++ std::map data structure, which maps the process id to the respective bounding box. This facilitates the bounding box set communication and the filtering in the receiving partition [57].

In the second level, ranks of B send their mesh partitions directly to the partner processes in solver A (step I in Fig. 2.6) using the point-to-point communication channels established in the first level. The processes of the receiving solver A compare their mesh partition to the received partitions to filter and identify the list of data that must be communicated during run time (step II in Fig. 2.6). The mesh filtering is done according to the configured data mapping scheme. Each process of A keeps only those vertices of the received mesh that influence the values at the target point. For example, in case of using the nearest neighbor mapping, only the closest data point (of the received mesh) to each data point at the target mesh is preserved and the other data points are filtered out. The filtering process is described in detail in [57, 51].

With the new scheme, the gathering of the complete coupling mesh data at the master process and the subsequent scattering at the partner solver can be avoided. Thus, it removes the memory issue of the previous approach (see Sec. 2.1) and the user can run highly parallel high-resolution simulations even if the whole coupling mesh does not fit into the memory assigned to a single process. In addition, the direct point-to-point mesh communication between partner processes improves the scalability of the initialization scheme.

2.2.2 Bounding Boxes for Parallel Filtering of Shared Mesh Vertices

In case of using RBF mapping, a parallel PETSC matrix [63] for the underlying linear system of the mapping has to be populated. In this matrix, every vertex of the input mesh represents one matrix line and every vertex of the output mesh a matrix column. To assemble this matrix correctly and efficiently, two steps are

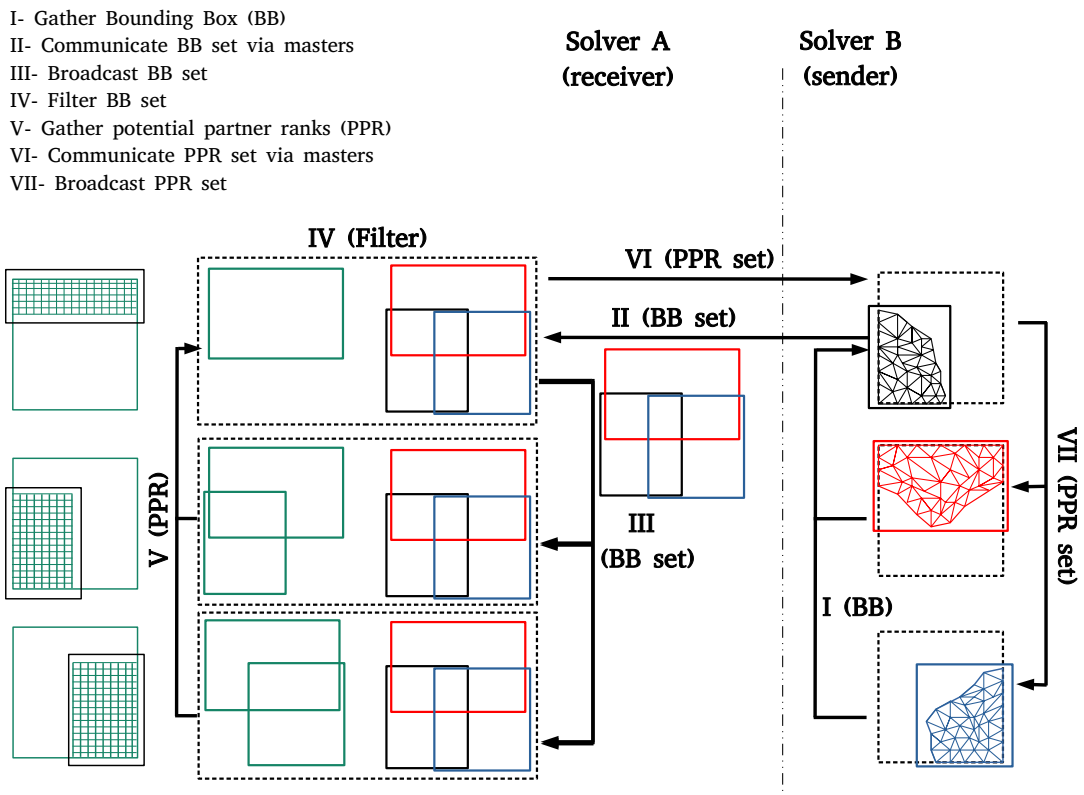


Figure 2.5: Two-level initialization scheme: the first level exchanges bounding boxes and establishes the communication channels between partner processes. The master process of B gathers the bounding boxes from other processes (I) and communicates them to the master process of solver A (II). The master process of solver A broadcasts the received bounding boxes to all other processes of solver A (III). Each process of A compares the received set of bounding boxes to its own to find the potential partner ranks (PPR) in solver B (IV). The complete set of sent bounding boxes is drawn in black, while the green boxes represent the subset relevant for the respective process of solver A. The list of PPR for all processes of A is gathered in the master process (V) and is communicated to the processes of solver B via master communication (VI) followed by a broadcast (VII), such that not only the processes of solver A, but also the processes of solver B know their potential communication partners [57].

required: (i) computing the pattern of the sparse matrix and allocating the respective PETSC matrix structure in a single step instead of allocating one entry after the other, (ii) computing the actual matrix entries. For both steps, it is necessary that each vertex is represented only once. Thus, RBF mapping requires completely separated

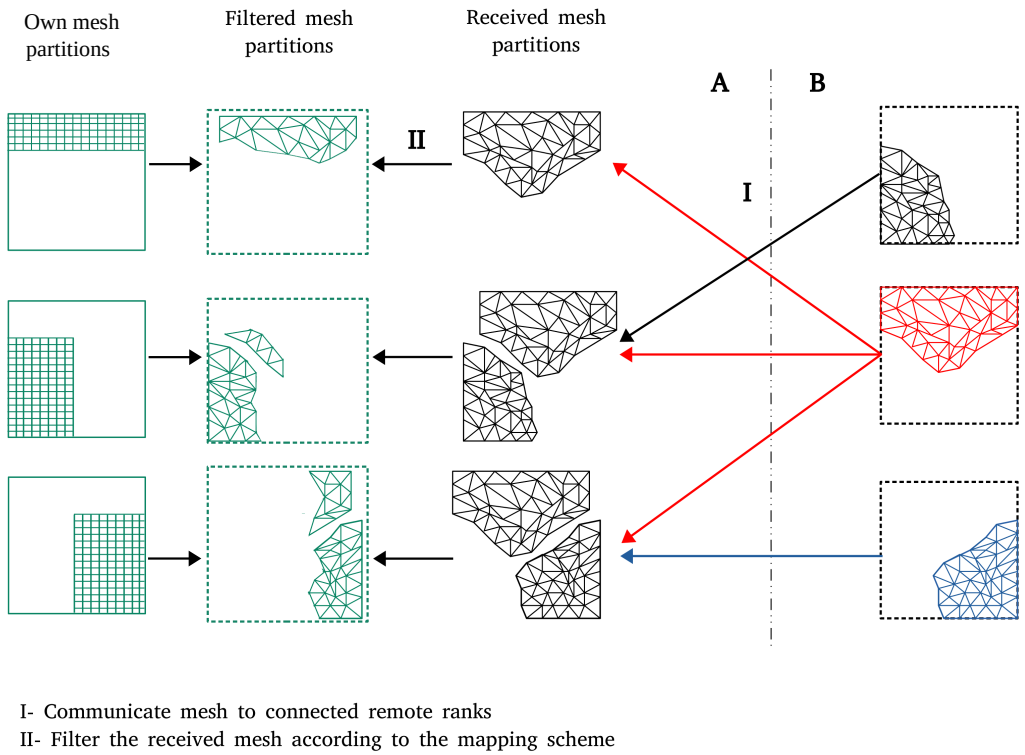


Figure 2.6: Two-level initialization scheme: the second level exchanges mesh partitions between partner processes to identify the exact list of data that have to be communicated during the simulation. Each process of solver B directly communicates its mesh partition to the relevant partner processes of solver A (I) using the channels established in level I. Each process of solver A compares its own mesh partition to the received mesh partitions and identifies the list of data that must be communicated during run time (II). The complete received mesh partitions are drawn in black and the parts that actually have to be communicated in green [57].

vertices lists, i.e. one vertex can not be shared among several ranks. This requires an elimination of duplications taking into account that the subsequent parallel solving step via preCICE requires a balanced distribution of matrix rows among the ranks. Therefore, the list of vertices must be checked after filtering to handle the shared entities. Previously, preCICE used to gather the list of owned vertices from the received mesh in the master rank, after filtering within individual ranks, to check and re-distribute the common entities. This procedure is inefficient as it exerts high load on the master rank while it idles the others. In the new approach, bounding boxes are used to parallelize the procedure and balance the load among all ranks.

Initially, each rank of the receiving solver must identify the neighboring ranks of the same solver before any decision about the shared vertices can be made. To do this, each rank of solver A computes a bounding box around its mesh partition (this refers to the own mesh and not the one that has been received from the other solver). These bounding boxes are gathered and broadcast via the master rank. Each rank identifies the neighbors by comparing its bounding box against the received set. The procedure is schematically shown in Fig. 2.7.

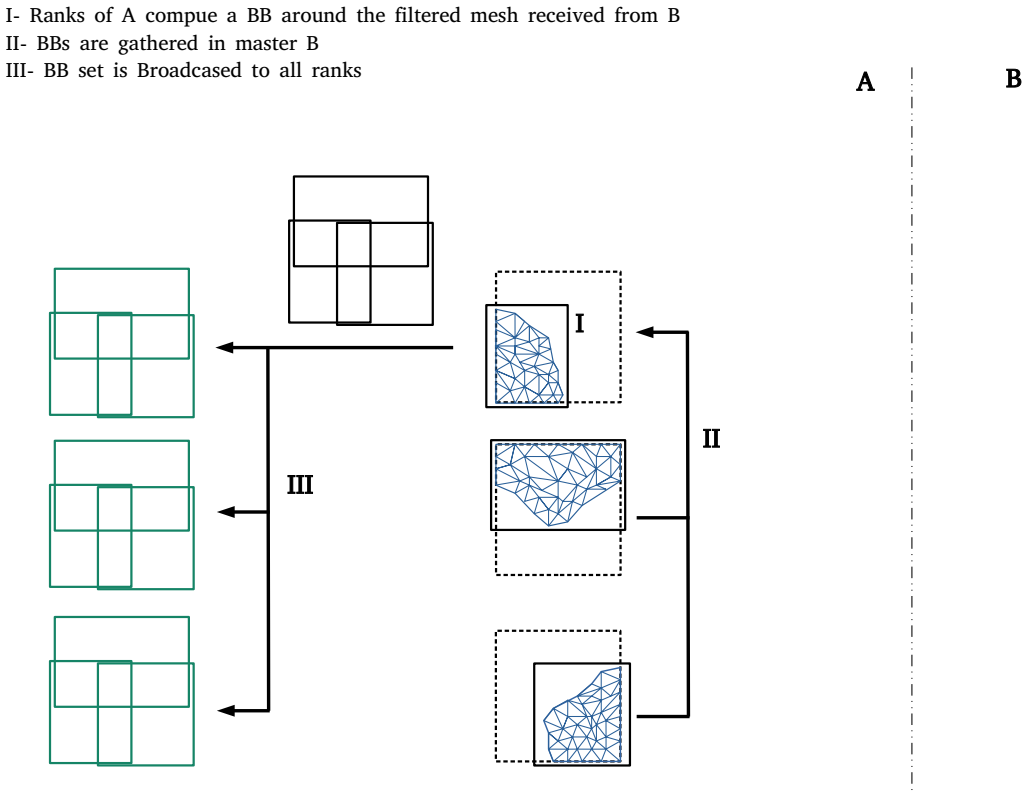


Figure 2.7: Two-level initialization scheme: The parallel filtering is aimed to filter all shared vertices among ranks of solver A to ensure that each vertex is tagged as 'owned' by only one rank. All ranks of A compute a bounding box around their mesh partition (I). The master process of A gathers the bounding boxes from other processes (II) and broadcasts them to the all processes of A (III). Each process can identify its neighbors by comparing the received bounding boxes with its own.

In the next step, each rank checks if vertices in its own partition are duplicated in neighboring partitions by testing if the owned vertices (in the mesh partition that is received from solver B and filtered according to the selected mapping scheme)

fit into the bounding box of a neighbor rank. Shared vertices along with the total number of owned vertices are then exchanged with the neighbors following a point-to-point direct communication. If a vertex is shared among **n** ranks, the following procedure is followed to decide on the owner of the vertex:

1. The rank with the lowest number of vertices owns the vertex.
2. If various ranks own equal number of vertices, the rank with the lowest id owns the vertex.

The above procedure is followed individually in all ranks. However, since all ranks have access to the list of shared vertices and number of vertices already owned by the neighbors, the procedure results in a consistent output.

2.3 Performance Results

We compare the old one-level with the new two-level initialization scheme of preCICE described in Sec. 2.2. To numerically demonstrate the improvements due to the new initialization approach, we present a test case, where we can evaluate the two-level initialization. A realistic coupling mesh is used along with two dummy solvers to only focus on the initialization phase. For performance comparison, an old version of preCICE (`1.5.2`) and a newer version (`2.2.0`) are used. Both were extended with additional time measuring commands and are available on the public preCICE git repository[1]. The performance analysis for the proposed scheme has been initially presented in [57]. In this section, I repeat the performance measurements and extend the analysis for a more profound explanation.

2.3.1 Test Case Description

We use the Abstract Solver Testing Environment (ASTE)[2] as the dummy solver, which is a framework that allows to imitate a parallel solver coupled via preCICE. This allows to inspect various performance characteristics of the preCICE initialization without the cost of using a real solver.

The test case setup involves two ASTE participants B and A. Each rank reads mesh

[1] `https://github.com/precice/precice/tree/performance-paper`
[2] `https://github.com/precice/aste/tree/mapping-tests`

data from given files and hands it over to preCICE by calling the preCICE Application Programming Interface (API). Participant B reads its mesh and provides both the mesh structure and the physical data to preCICE. Participant A reads its mesh and provides the mesh structure to preCICE. A then receives both the mesh and the data from participant B, uses the configured consistent mapping to map the received data to the local mesh, and finally writes the resulting vertex data to a file. The overview of this process is depicted in Fig. 2.8.

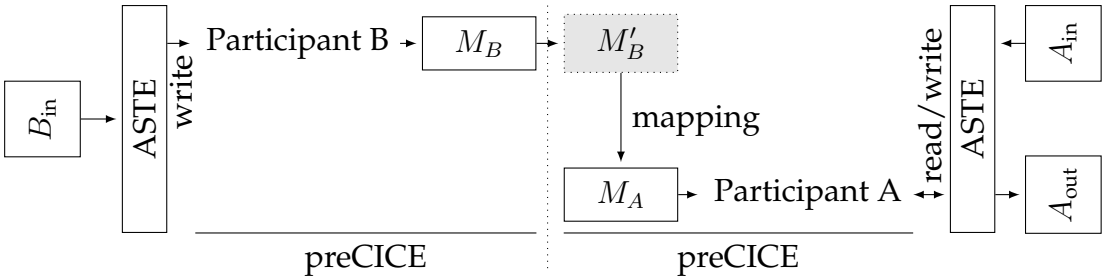


Figure 2.8: Test configuration using ASTE. Participant B on the left reads the mesh structure and physical data from a file B_{in}. Participant A on the right reads the mesh structure from the file A_{in}. B and A initialize communication, then data are transferred from M_B to M'_B, mapped to M_A using a nearest-projection mapping and finally written to a file A_{out}. This illustration is taken from [57].

We use surface meshes of a turbine blade geometry[3] shown in Fig. 2.9 for our numerical investigations. The geometry was triangulated using GMSH[64] fixing the edge-length to achieve an almost uniform element size and shape.

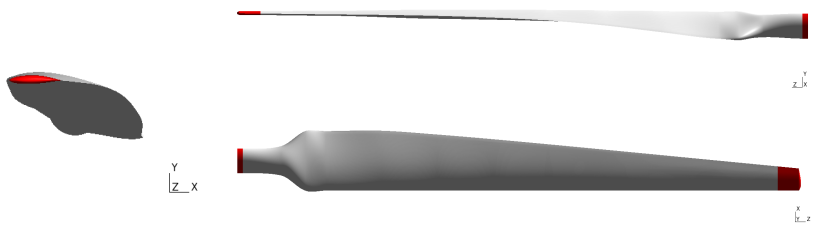


Figure 2.9: Different perspectives on the turbine-blade test geometry. This illustration is taken from [57].

For inter-code data exchange, we use TCP/IP socket communication. For data mapping between non-matching interface meshes, we use the nearest projection scheme

[3]Wind Turbine Blade created by Ivan Zerpa, February 7th, 2012 `https://grabcad.com/library/wind-turbine-blade--4`

provided in preCICE. In addition, both the overall initialization times and more detailed breakdown values are reported per core average. For all experiments, we avoided synchronization to minimize times and, thus, the initialization time. Though it optimizes the overall initialization time, may slightly reduce the communication-related events breakdown accuracy due to events' time overlaps.

2.3.2 Performance Analysis

To show the complexity and scalability improvements in preCICE, we conduct both strong and weak scaling studies. All measurements are carried out on the SuperMUC-NG supercomputer[4] at the Leibniz Supercomputing Centre of the Bavarian Academy of Science and Humanities (LRZ). This machine consists of 3.1GHz Intel Xeon Platinum 8174 (SkyLake) processors. Each node contains two processors with 24 cores per processor (48 cores per node) and 96GB of RAM. The nodes are connected via Intel Omni-Path interconnect.

Strong Scaling

The strong scalability of the developed initialization scheme is evaluated using various computational meshes, which are given in Table 2.1. Mesh M4 (with mesh width 0.005 and $628\,898$ vertices) is used to compare the overall performance of the newly developed two-level initialization and the old one-level scheme since this is the largest mesh that can be handled by the old version of preCICE. The one-level scheme is unable to handle finer meshes due to the memory issues explained in Sec. 2.1. On the other hand, the new two-level initialization scheme would be applicable for much larger meshes as the memory bottleneck induced by sending the complete coupling meshes from A to B via master communication has been eliminated. Accordingly, we also present strong scalability measurements with finer meshes only for the two-level scheme later in this section.

Figure 2.10 compares the initialization times of both versions using mesh M4. The number of CPU cores indicates the total number of processes, i.e., MPI ranks for both domains together. The available cores are divided between the domains with a ratio 1:3 to ensure that we do not get matching partitions between both solvers.

[4] `https://doku.lrz.de/display/PUBLIC/SuperMUC-NG`

Table 2.1: Strong scalability study: Meshes for the wind turbine blade at varying mesh resolution (mesh width). The mesh width indicates the average edge length used to construct the surface mesh [57].

Mesh ID	Mesh width	Number of Vertices	Number of Triangles
M4	0.0005	628 898	1 257 391
M5	0.0004	1 660 616	3 321 140
M6	0.0003	2 962 176	5 924 260

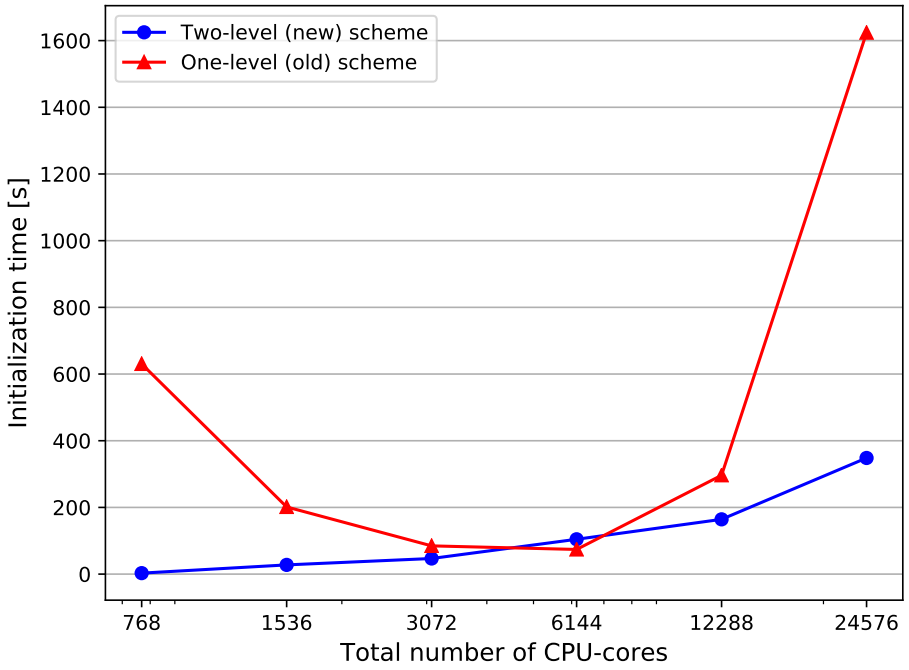


Figure 2.10: Strong scalability measurements: Total initialization time comparison between the two-level approach and the previously used one-level scheme. A mesh with mesh width 0.005 resulting in 628 898 vertices (Table 2.1, M4) is used for conducting the analysis. This illustration is taken from [57].

The comparison proves significant initialization time reduction compared to the one-level scheme. While we observe a better performance for the one-level scheme for the cases with a small number of CPUs, the new scheme significantly outperforms the old one for large numbers of CPU-cores. The initialization time for 24,576 CPU-cores is less than 6 minutes for the two-level scheme, while the old scheme requires approximately 5 times more time. All CPU-cores exploited for this experiment are located at the common interface. In a real surface-coupled simulation, the interface ranks are only a small fraction of the total ranks. Therefore, the introduced scheme

is expected to be very efficient even when a simulation exploits the whole machine capacity (several hundred thousands of CPU-cores).

The observed improvement is attributed mainly to (i) replacing the gather-scatter mesh communication with the bounding box scheme and (ii) enhancing the nearest projection implementation in preCICE. While the former is discussed here, details about the latter can be consulted in [57]. Fig. 2.11 compares the boundary mesh communication time between the one-level and two-level initialization schemes. In the mesh communication phase, coupled solvers exchange their interface mesh partitions (Fig. 2.6 step I for the new two-level scheme, Fig. 2.4 steps I to III for the old one-level scheme). The received mesh partition is filtered and the interpolation is computed in the mapping computation and depicted in Fig. 2.6, step II. The comparison indicates, that the new scheme has significantly reduced the required time for the mesh communication. This reduction is due to the replacing the (via-)master mesh communication in the old scheme with a point-to-point method in the new one (see also Fig. 2.12). Figure 2.11 shows a strong increase of the communication time with increasing number of cores in the one-level scheme for the mesh communication. The mesh communications in the old scheme consists of a sequential gathering of mesh partitions from all processes (in a loop over all processes, see Fig. 2.4 step I), communicating the whole mesh partition at the common interface via master communication (Fig. 2.4 step II) and broadcasting it to all processes in the receiving solver (Fig. 2.4 step III). The sequential gather and broadcast are $O(p)$ in the number p of the processes. Therefore, the increase in the mesh communication cost is expected. The mesh communication cost for the new scheme is analyzed later in this section.

In the following, we focus on further analyzing the run time and scalability of the new two-level approach in a further breakdown study. The breakdown of the complete initialization time is given in Fig. 2.12.

The figure shows, that the bounding box comparison and feedback (Fig. 2.5, steps II and III) constitute the majority of the run time for the new two-level initialization, in particular for core counts larger than 1536. The increase in the time spent for bounding box comparison for a higher number of processes is expected since for a higher number of processes in the partner solver, each process needs to compare its bounding box with more partner processes. Currently, this operation is O(p^2) in the number p of processes in total and O(p) per process. In addition, the time required for the feedback phase, which consists of gathering feedbacks from processes in the master rank and sending it to the other solver also increases with the number of

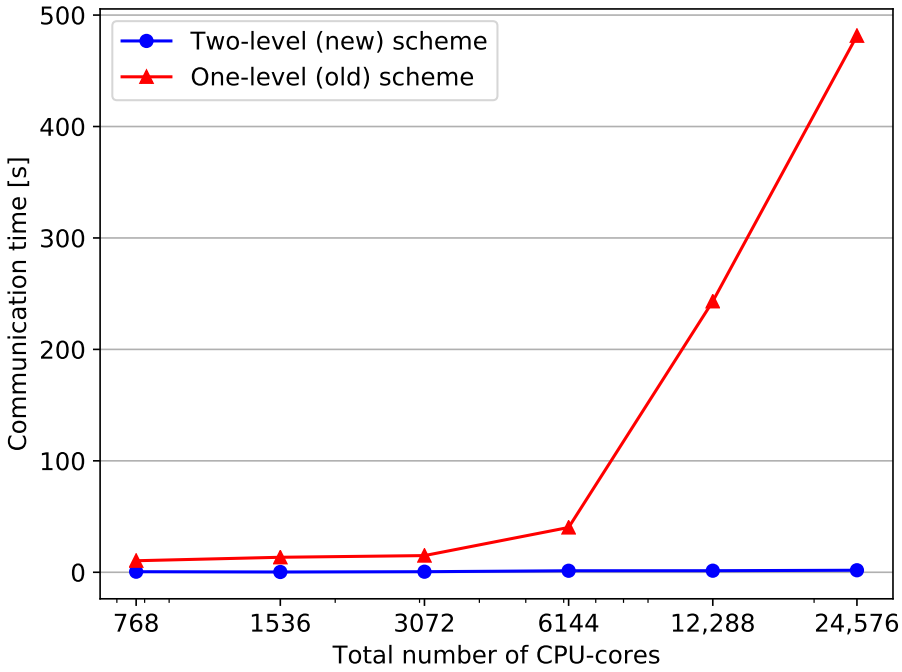


Figure 2.11: Strong scalability study: Comparison of run times for the mesh communication between between the two-level and the one-level scheme. The mesh M4 is used for conducting the analysis. This corresponds to Fig. 2.6 step I for the two-level scheme and Fig. 2.4 steps I to III for the one-level scheme. This illustration is taken from [57].

processes. This increase is also expected, as the higher number of processes includes more communication in the gather operation.

We observe from Fig. 2.12, that the time required for mesh communication decreases with the increasing number of cores up to 1536 and increases afterward. The reduction is due to the decreasing size of mesh partitions, that are communicated, while the latter increase might be because of the higher communication overhead. This can be due to an unfavorable distribution of the communication partners on the machine or due to the higher accumulated number of communication events within the communicator. However, this step takes only about 1s and its contribution to the total initialization time is very small.

The time to communicate bounding boxes also increases with the increasing number of cores. This is because, for higher core counts, more bounding boxes are gathered in the master rank, communicated via the master ranks, and broadcast to the slave ranks of the other solver (Fig. 2.5, steps I to III). This operation is inevitable in the current approach. However, its contribution is very small (less than 100 ms).

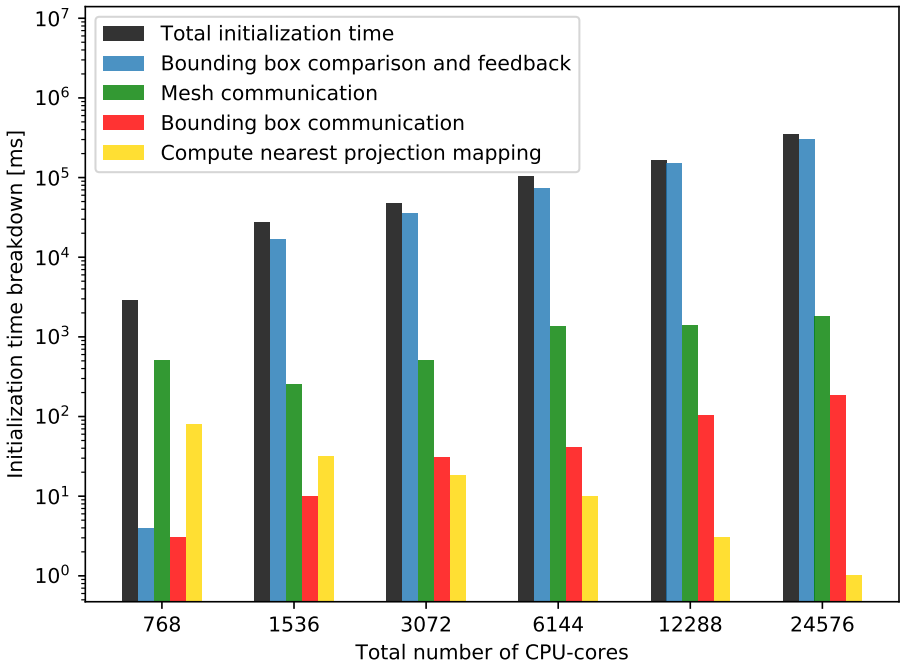


Figure 2.12: Strong scalability study: Initialization time breakdown for the two-level initialization approach using mesh M4. Only parts significantly contributing to the run time are depicted: 1- Bounding box comparison and feedback (Fig. 2.5 steps IV to VII). 2- Mesh communication (Fig. 2.6 steps II). 3- Bounding box communication (Fig. 2.5 steps II and III). 4- Compute nearest projection mapping (Fig. 2.6 step II). This illustration is taken from [57].

Finally, the run time for the mapping computation (Fig. 2.6, steps I) decreases with the increasing number of cores. In this step, it is computed that how the values in the mesh points of the partner solver must be interpolated to the mesh points of the own mesh. Increasing the number of cores reduces the size of the mesh partitions that must be compared to compute this mapping, thus the comparison becomes cheaper. Therefore, the reduction in the mapping computation time corresponds to our expectations. In addition, recent improvements to the mapping computation in the preCICE library strongly contribute to this enhancement. More details can be found in [57, 34].

For a deeper analysis of the performance and efficiency of the two-level scheme, we present the initialization time breakdown in Fig. 2.13 for the case that coupling solvers use non-matching meshes. For this experiment, we use mesh M5 for solver A and M6 for solver B.

The figure shows, that, even for a finer mesh, the bounding box comparison and

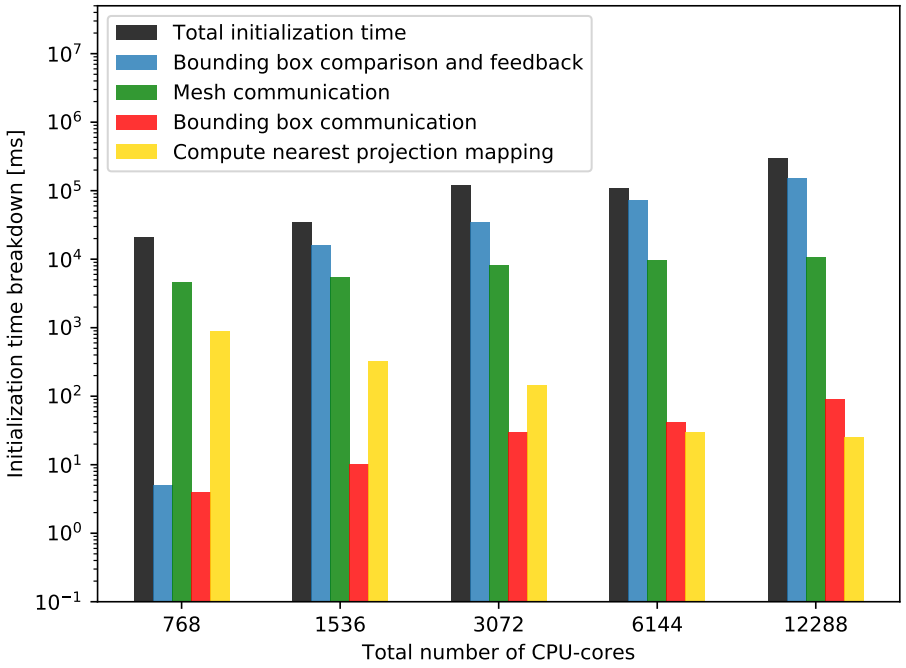


Figure 2.13: Strong scalability study: Initialization time breakdown for the two-level initialization approach using mesh M5 for solver A and mesh M6 for B. Only parts significantly contributing to the run time are depicted: 1- Bounding box comparison and feedback (Fig. 2.5 steps IV to VII). 2- Mesh communication (Fig. 2.6 step II). 3- Bounding box communication (Fig. 2.5 steps II and III). 4- Compute nearest projection mapping (Fig. 2.6 step II). This illustration is taken from [57].

feedback are still the most expensive components. The time spent for bounding box communication also increases with increasing number of cores. However, this cost does not depend on the mesh size and is only correlated with the number of interface processes, as expected. In addition, we observe a gradual increase in the mesh communication time, which is probably due to a larger communication overhead for the cases with higher core number and, thus, an increasing overall number of messages. The mesh communication time is higher than for the case with mesh M4, which is due to the larger mesh. Similar to the previous case with smaller mesh, the mapping computation time is still insignificant compared to the other components.

Weak Scaling

To better analyze the performance of the new initialization scheme, including memory efficiency, we present weak scalability measurements in this section. For this

purpose, the distribution of CPU cores among the solvers A and B for all used meshes are specified in Table 2.2. As seen, the mesh size per partition is kept almost constant for both solvers to satisfy the required condition for the weak scaling analysis.

Figure 2.14 compares the initialization time of the new two-level approach and the old scheme for increasing mesh sizes and increasing the total number of cores. The measurements for the old initialization scheme are restricted to meshes M1–M4, since it runs out of memory for meshes with higher resolution, as explained in Sec. 2.1.

Table 2.2: Weak scalability study: Meshes for the wind turbine blade at varying mesh resolution (mesh width). The mesh width indicates the average edge length used to construct the surface mesh. The total number of cores is approximately proportional to the number of mesh vertices. The available CPU cores are distributed with a 1:3 ratio between the solvers [57].

Mesh	Mesh	#Vertices	Cores			#Vertices per core	
ID	Width	Total	Total	B	A	B	A
M1	0.0025	25 722	104	26	78	989	330
M2	0.0010	165 009	720	192	528	859	312
M3	0.000 75	330 139	1344	336	1008	982	328
M4	0.0005	628 898	2496	624	1872	1007	336
M5	0.0004	1 660 616	6144	1536	4608	1081	361
M6	0.0003	2 962 176	12 288	3072	9216	964	321

An excellent reduction in initialization time is observed when replacing the old method with the new two-level scheme. Figure 2.15 shows, that the new scheme significantly reduces the time for the communication of the interface mesh. In addition, the new scheme is capable of handling very large interface meshes, since it does not use any central mesh communication instance. Even though the initialization time for the finest mesh M6 and 12288 CPU cores is still in the range of a few minutes, we observe a steep increase for larger meshes, which is due to the increase in mesh communication time as seen in Fig. 2.15.

To analyze the reason of the mentioned performance drop, we present the initialization breakdown for various mesh sizes and CPU cores in Fig. 2.16. We observe, that the mesh communication time increases considerably for meshes M5 and M6. This increase might be due to the higher accumulated communication events and accumulated data size (which is communicated) for higher mesh resolutions and higher number of processes. However, deeper investigations are necessary to disclose the cause of this behavior. The bounding box communication and comparison cost is the dominant component also in this example. This cost only depends on the total num-

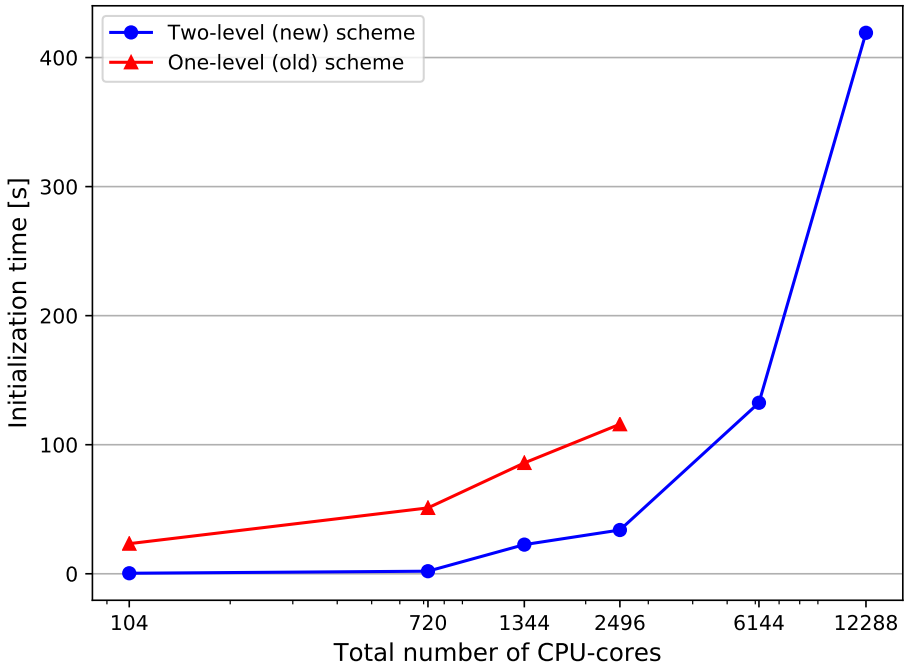


Figure 2.14: Weak scalability study: Total initialization time comparison between the two-level and the one-level schemes. The core distribution and the mesh information are given in Tab. 2.2. This illustration is taken from [57].

ber of cores, not on the mesh resolution as can be seen in a comparison of Fig. 2.16 and Fig. 2.12. For the mapping computation, we observe an almost constant time, which is expected as a result of the constant mesh partition size per core. Therefore, the super-linear increase in the total initialization time can be mainly attributed to the bounding box communication and comparison and the mesh communication.

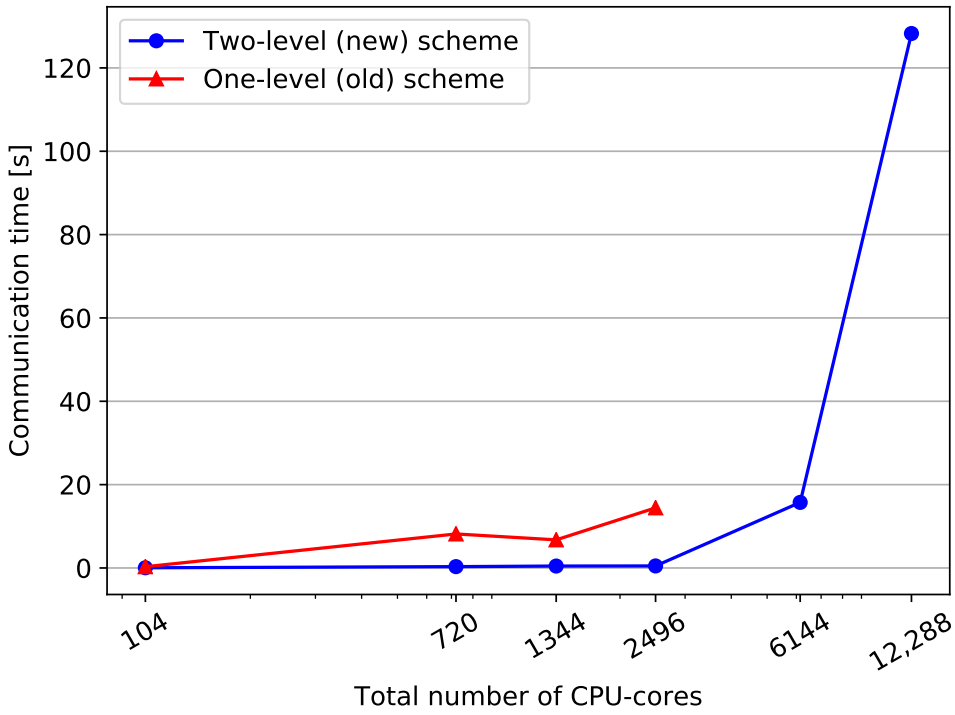


Figure 2.15: Weak scalability study: Comparison of mesh communication time between the two-level and the one-level scheme. The core distribution and the mesh information are given in Tab. 2.2. This illustration is taken from [57].

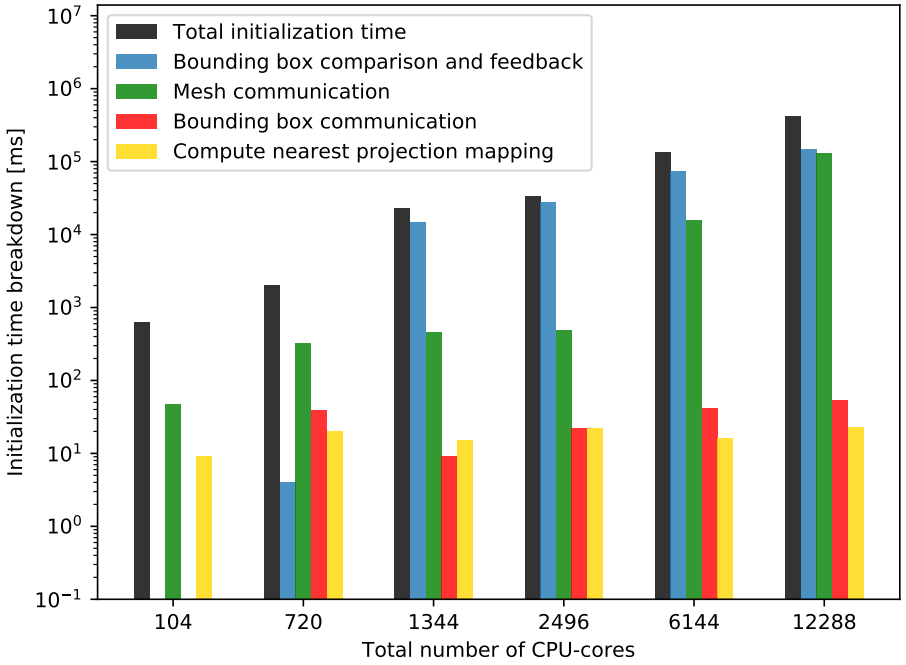


Figure 2.16: Weak scalability study: Initialization time breakdown for the two-level initialization scheme. Only algorithmic parts with significant contributions to the initialization run time are depicted. : 1- Bounding box comparison and feedback (Fig. 2.5 steps IV to VII). 2- Mesh communication (Fig. 2.6 step II). 3- Bounding box communication (Fig. 2.5 steps II and III). 4- Compute nearest projection mapping (Fig. 2.6 step II). The core distribution and the mesh information are given in Tab. 2.2. This illustration is taken from [57].

2.4 Summary

In this chapter, it is explained how the initialization of the coupling library preCICE is improved by replacing the previous one-level scheme with a performant two-level scheme. This improved the scalability of the initialization by replacing gather-scatter mesh communication with a parallel point-to-point scheme. In addition, it removed the memory bottleneck of the one-level scheme which was due to gathering interface mesh in a master rank.

To evaluate the proposed algorithms, we presented strong and weak scaling measurements for an artificial turbine blade test case using various mesh resolutions. The performance analysis showed, that five times faster initialization can be achieved with the new implementation on $24,576$ (interface) cores using a coupling interface mesh with $628,898$ vertices. By avoiding the memory bottleneck of the old one-level (gather-scatter) scheme, the new method is able to handle much larger meshes. The largest surface mesh which is considered in this chapter had nearly three million vertices, which, in a real simulation, could correspond to a coupled setup running on a complete supercomputer (several hundred thousand cores). The new method is able to initialize this setup in less than six minutes.

Detailed performance analysis revealed, that the most expensive component of the new scheme is the comparison of bounding boxes and the sending of connection feedback to the coupling partner. Using more efficient data structures such as linked-cells to store the set of bounding boxes could potentially further improve the initialization.

3 Data-Driven Performance Modeling and Load Balancing

This chapter is concerned with the inter-solver load balancing in large-scale partitioned multi-physics/multi-scale simulations where separate solver codes are used for different physical phenomena and additional software for technical and numerical coupling. An unbalanced distribution of computational resources among single-physics solvers can lead to idle times in one or multiple solvers (in case more than two solvers are coupled) and can drastically reduce the computational performance of the coupled simulation. A data-driven approach is developed to address this issue and improve the performance of the simulations. For this purpose, two different methods are considered for the empirical performance modeling of single-physics solvers: (i) the so-called performance model normal form (PMNF) regression and (ii) neural networks. Based on these performance models, an appropriate optimization problem is derived and solved to find the optimal distribution of computational resources between solvers. The optimization problem directly depends on the equation coupling type (serial or parallel). We present two different test cases to evaluate the effect of the proposed method. For both test cases, we use the Ateles solver provided by the APES framework [65]. Furthermore, the preCICE library [34] is used to couple the subdomains of the partitioned solver. Performance analyses show that the proposed method provides significant improvements in terms of load balancing and scalability. In most cases, the proposed method can almost remove the load imbalance completely, hence the run time is decreased considerably (by up to 25%) and the load imbalance is reduced to around only 1%. The remainder of this chapter is organized as follows. Section 3.1 introduces the load balancing issue for the partitioned multi-physics problems. The empirical performance modeling methods are explained in Sec. 3.2. This includes both a single-variable model that uses only the core number as the input and two multi-variable models that require the problem size as well. The derivation of the optimization problem and its solution to calculate an optimal computational resources distribution are presented in Sec. 3.3. We present a multi-step load balancing approach for the test cases that use large computational mesh, where we do not have access to enough performance data to build cheap multi-variable performance models. Section 3.5 describes a load balancing method developed specifically for the APES framework. We compare the results of

the proposed general method to this specific approach. Section. 3.6 provides performance analysis results for real-world applications and Sec. 3.7 summarizes and concludes the chapter.

3.1 Introduction to Load Balancing in Multi-Physics Simulations

The simulation of multi-physics and multi-scale problems is a challenging task in terms of the realization, physical accuracy, and computational efficiency, in particular when the scalability of the solution on massively parallel computers is concerned. As explained in chapter. 1, using efficient single-physics simulation codes does not automatically guarantee to achieve a good parallel efficiency for a partitioned coupled multi-physics simulation. One of the issues that multi-code coupling introduces is the load balancing between solvers. This type of load balancing is different from intra-solver load balancing. Depending on the equation coupling scheme between domains (see Sec. 1.1 for details), solvers may require boundary data from each other at the beginning of each time step and/or an iteration. This is where the load imbalance and/or performance drop can occur. For example, in the case of a parallel coupling (implicit and explicit), if one of the solvers is slower than the other one, the faster solver will have to wait for the input from the other one. This can idle the faster solver and significantly reduce the performance of the coupled framework. Since the coupled solvers use different degrees of hardware optimizations, different numerical methods, unpredictable numbers of iterations per time step and equation systems, etc., it is impossible to use analytical or heuristic weights per discretization point for balancing the load across solvers. We propose a novel, data-driven two steps approach to address the issue. The proposed method requires a performance model for each solver that can estimate its simulation run time as a function of the number of used CPU cores. Then, an integer optimizer is needed to use the performance models to calculate the best distribution of CPUs among solvers. This optimization depends mainly on the type of coupling. This chapter explains the proposed load balancing approach for both serial and parallel coupling schemes and investigates sophisticated test cases exploiting large supercomputers. In addition, to prove the method's capability, we compare the results to a solver-specific scheme and show that the presented method can distribute the computational resources with the same performance, even better in some cases, and with a lower overhead cost. Note that the proposed method is suitable for black-box partitioned approaches that use independent solvers for each sub-problem and incorporate a library to couple

them. In this approach, one `mpiexec` command per solver is executed which pins an independent set of `MPI` processors to each solver. Therefore, processors can not be shared between solvers and, thus, must be distributed among them..

3.2 Performance Modeling

To properly distribute available computational resources among solvers and achieve the optimal load balancing, a performance model for each solver is necessary that can predict single physic solvers' run time. Due to the complexity of the solvers' performance, using analytical models is not feasible and thus, we use empirical models instead. We introduce and adapt two methods for this purpose: (i) the so-called performance model normal form (PMNF) regression [66] and (ii) neural networks. Although PMNF regression shows promising results for single variable (number of cores) models, the results for multi-variable models (number of cores and problem size) are not very satisfactory. Therefore, we introduce an alternative approach using neural networks. Both methods need some performance measurements as input to build the desired model. These measurements are obtained by running coupled simulations for a few time steps. All models map a set of parameters $v_1, \ldots, v_d$ to the predicted run time. We aim to determine the mapping function f such that it minimizes the loss function $L(f(v_1^k, \ldots, v_d^k), y^k)$, a suitable distance measure between predicted run times and measurements y^k for m sets of parameters $v_1^k, \ldots, v_d^k, k = 1, \ldots, m$. The performance modeling process, which we use in the current work, is schematically shown in Fig. 3.1. The performance modeling using a single input parameter (number of processes) has been described in [67, 68]. This section extends the idea for a multi-parameter modeling.

3.2.1 PMNF Regression

The PMNF regression was originally introduced for the performance modeling of code kernels to find and resolve scalability issues [66]. In this chapter, we use this regression to model a whole physical solver's performance. The model maps p number of cores (CPUs or GPUs) assigned to each solver to its run time. The aim is to find the mapping function f that minimizes the loss function $L(f(p^k), y^k)$, a suitable distance measure between predicted run times and measurements y^k, where k is the number of performance measurements. For the modeling, we initially do a

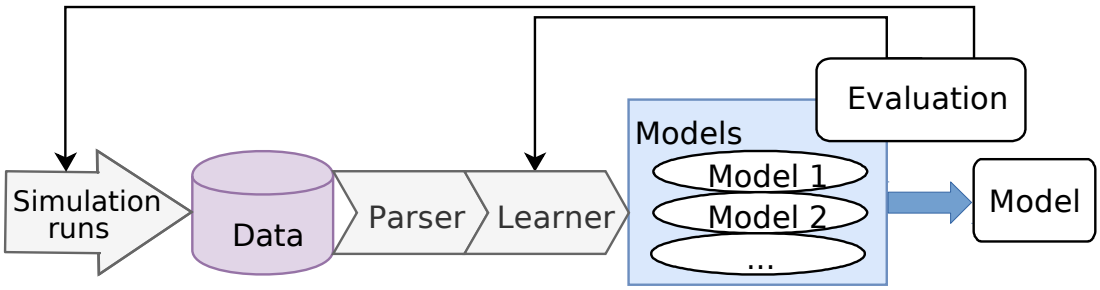


Figure 3.1: Performance modeling for black box load balancing across solvers: workflow to find the best matching performance model. The performance models are checked by metrics evaluated on a validation data set to select the model with the smallest validation error. This illustration is taken from [68].

few performance measurements to gather the required data for the regression. The authors of [66] suggest that only a few data points are sufficient for an accurate model. Later in this chapter, we will show that with 6 measurements (5 for training and 1 for validation) we can accurately model the solver's performance. Calotoiu et al. [66] suggest the following formulation for the run time model:

$$f(p) = \sum_{k=1}^{n} c_k p^{i_k} \log_2^{j_k}(p) \tag{3.1}$$

where i_k, j_k, and n are the model hypotheses, and the coefficients c_k are the degrees of freedom of the regression. Calotoiu et al. [66] suggest that the search space given by $n = 2, i_k \in I = \{0, \frac{1}{4}, \ldots, \frac{12}{4}\}$ and $j_k \in J = \{0, 1, 2\}$ is sufficient for many applications. However, our investigation shows that extending the search space to $n = 2, I = \{0, \pm\frac{1}{4}, \ldots, \pm\frac{12}{4}\}$ $J = \{0, \pm 1, \pm 2\}$ improves the modeling. We use the mean square error (MSE) between the measured and the predicted run times as a loss function. To find the optimal model, we simply check all hypotheses within the search space, calculate a cross-validation-based loss, which is the accumulative error on a validation data set, for each model, and pick the one with the smallest validation loss. This process must be done for all solvers to find the appropriate performance model for each of them [68].

3.2.2 EPMNF Regression

Providing enough performance measurements for PMNF regression can be expensive, for instance when a very large computational mesh is used for a simulation. In such cases, one possible remedy is to use a multi-variable performance model which uses both the core number and the mesh size as an input to predict the run time. This allows building the model with performance measurements gathered from much smaller mesh sizes.

The PMNF regression is, however, limited to a single input parameter modeling (the number of ranks assigned to the solver). Calotoiu et al. introduce the extended performance model normal form (EPMNF) to include more variables in the performance model. For d parameters $v_1, \ldots, v_d$, the EPMNF is given by [66]:

$$f(v_1, \ldots, v_d) = \sum_{k=1}^{n} c_k \prod_{l=1}^{d} v_l^{i_{k,l}} \log_2^{j_{k,l}}(v_l). \tag{3.2}$$

Where n is the number of regression terms for each variable. If we use only two terms for each variable in the regression, we need to find $2n \times d$ parameters in total. Analogous to the one dimensional case, we define two sets I and J of possible values, i.e., restrict our search to $i_{k,l} \in I$ and $j_{k,l} \in J$.

The large search space of this model quickly becomes a problem. In the single parameter case, we have to check $|I| \cdot |J|$ possible combinations for each of the n terms. We do not need to consider models with repeated terms and their order does not matter, resulting in a total of $\binom{|I|\cdot|J|}{n} = \binom{25\times5}{2} = 7750$ combinations for the extended set of $n = 2, I = \{0, \pm\frac{1}{4}, \ldots, \pm\frac{12}{4}\}$ $J = \{0, \pm1, \pm2\}$. In the d-dimensional case, the number of parameters per term increases exponentially with d. There are a $(|I| \cdot |J|)^d$ possible combinations per term and $\binom{(|I|\cdot|J|)^d}{n}$ total models. Expanding the example search space to $d = 2$ dimensions leads to $\binom{(25\times5)^2}{2} = 122,062,500$ candidates, or $\binom{(25\times5)^3}{2} \approx 1.9e12$ for $d = 3$. Note that the evaluation of each hypothesis is not trivial, since we need to generate data, calculate coefficients, and evaluate the cross validation loss for each. In conclusion, generating and comparing all models in the search space is generally not feasible, except for very small search spaces.

Calotoiu et al. [66] propose a heuristic approach that drastically accelerates the generation process for multi-dimensional models. **Hierarchical search** prunes the search space to only include hypotheses that are combinations of the best single parameter models. This approach aims to reduce the number of evaluated candidates by selecting only those model hypotheses which are likely to be the best ones. Accord-

ingly, we determine the best single parameter models in a first step based on data generated with representative values for the other parameters, build all combinations, generate a multi-dimensional data-set varying all parameters, and choose the optimal models for this reduced set as described above, i.e., we then determine the cross-validation based loss for both of these and choose the one with the smaller loss. Note that there is no guarantee that the best model of the full search space is, in fact, part of the restricted one.

If $n > 1$, we need to explore all possible options that can be obtained by combining all subsets of terms. There are 2^n such subsets for each of the d single parameter models, resulting in a total of 2^{nd} combinations. To better understand the impact, that it has on the number of evaluated hypotheses, let us look at our running example $n = 2, I = \{0, \pm\frac{1}{4}, \ldots, \pm\frac{12}{4}\}$ and $J = \{0, \pm 1, \pm 2\}$ and $d = 3$: In order to find the three best single parameter models, we need to check $3 \times 7750 = 23,250$ hypotheses and then compare the $2^{2\times 3} = 64$ possible combinations, which makes for an overall number of $23,314$ regression problems, reducing the full search space of $1.2e12$ models to less than 0.0000002% of its original size.

Later in Sec. 3.6, we show that the EPMNF method with hierarchical search is not able to predict the solver performance accurately in a test case with the input parameters core count and problem size. In addition, it is not easily possible to increase the number of input parameters, e.g., include the hardware properties in the performance model. We have shown that the complete search space for two variables is already very big and adding more variables extends the search space even more. Therefore, we take an alternative approach and use neural networks (NNs) to model the solver performance against the number of cores and the problem size. Studying the effect of hardware properties on the solver's performance is out of this chapter's scope and is a topic for future research.

3.2.3 Neural Networks

After discussing the drawbacks of EPMNF for multi-variable performance modeling, we aim to use neural networks to predict the run time of the solvers involved in the coupled simulation and study the effect of different parameters, e.g., problem size and core counts, on the run time. Deep learning networks have become more popular in the past years and they have been tried out for many problems. This popularity is mainly due to their success in pattern recognition and image classification which convinced scientists to try them out as a remedy for other applications. Although

neural networks were already introduced in 1943 [69], they have barely been used in practice for many years, since there was no efficient way to train them, computers were not powerful enough to handle large networks and available training data were insufficient. Small networks were not capable of solving the sophisticated problems and lack of computing power and sufficient training data hindered using larger networks. Therefore, the capability of this method remained unknown for many years. However, the development of more powerful computers and the introduction of the back-propagation algorithm changed the story. New algorithms and improved hardware made them very powerful choices for certain problems (e.g., image classification [70] and speech recognition [71]) in the early 2010s [72].

There exist many variants of neural networks, we focus on the densely connected feed-forward networks for the first step. A schematic illustration of a densely connected neural network with two hidden layers is shown in Fig. 3.2. Each hidden layer comprises several neurons, whose input-output relation is determined by an activation function $\sigma : \mathbf{R} \rightarrow \mathbf{R}$. The output values are then mapped to the input of the next layer via multiplication with a weight-matrix W. In addition, each layer has a bias term h. This can be interpreted as a neuron with a constant output of 1. Each layer's output can be calculated as $S^{out} = \sigma(W^i S^{in} + h^i)$ in the forward path, where S^{in} and S^{out} are the input and the output of the layer i. A set of training data is needed to train the neural network, i.e., to determine the entries of the weight matrices by minimizing a given loss function. Neural networks can be used to represent a mapping with arbitrary input and output dimensions. Since we study the solver's run time, considering NNs with one-dimensional output is sufficient. Various hyperparameters and methods were tried and the following setup is selected as it led to the best performance. We use leaky rectified linear units (ReLUs) as the activation function and the mean square percentage error (MSPE) between the predicted and the measured run time as the loss function. We use the Adaptive Moment Estimation (ADAM) [73] optimization method to minimize the loss function and update the weights and apply a dropout regularization technique to avoid over-fitting [74]. Each item is explained in detail in the following sections. In the current work, we use the Keras package with TensorFlow backend [75] to implement the neural network.

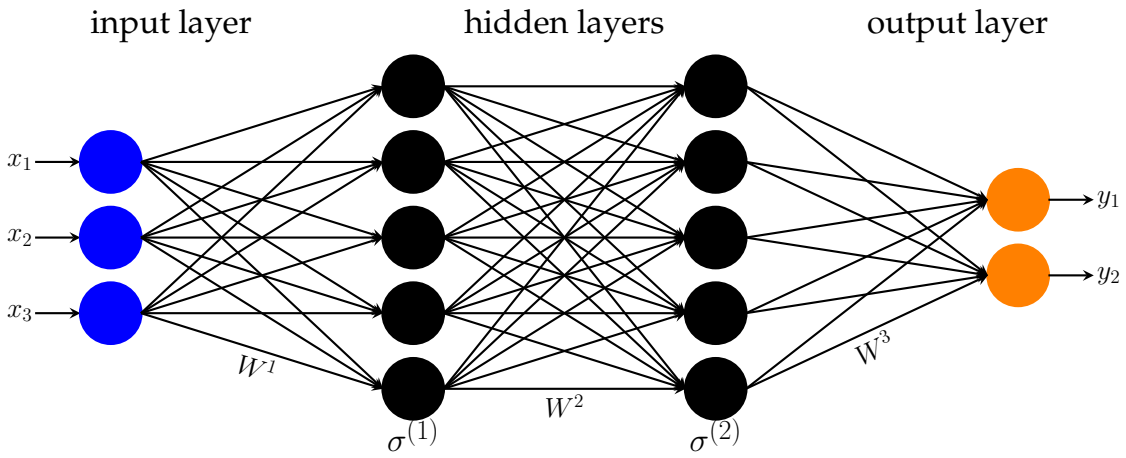


Figure 3.2: A schematic depiction of a neural network with 2 hidden layers, where x_i and y_i are the input and the output of the network, W^i represents the weight matrix and σ^i is the activation function of layer i.

ReLU Activation Functions

To define a neural network, a non-linear activation function σ is needed to transform the input of a neuron into its output. There are plenty of suggestions for possible functions with different properties and use cases. We tested different options and rectified linear units (ReLUs) performed best for our case.

Glorot et al. [76] showed that rectified linear units (ReLUs) have a better performance for training the NNs than sigmoid and hyperbolic tangent functions. Since 2017, they are therefore the most commonly used activation function in deep learning [77]. In the simplest case, σ is given by

$$\sigma(x) = \max(0, x). \tag{3.3}$$

Although the gradient is zero for all negative inputs, it does not vanish for large x (as it does for sigmoid activation, e.g.). A vanishing gradient stops the network's learning and weights do not update anymore. At $x = 0$, the gradient is undefined, but in practice, $x = 0$ very rarely occurs. Nonetheless, a value for this unlikely case should be selected, e.g., zero. The problem for negative inputs can be fixed with leaky ReLUs:

$$\sigma(x) = \begin{cases} x & \text{for } x \geq 0, \\ \varepsilon x & \text{for } x < 0, \end{cases} \tag{3.4}$$

where ε is some small value e.g. $\varepsilon = 0.01$. Alternatively, ε can also be considered as a parameter that must be learned alongside the weights[72]. We use a constant $\varepsilon = 0.01$ in the current work.

Adaptive Moment Estimation (ADAM) Optimizer

The most important factor for creating well-performing neural networks is the optimization of the entries of the weight matrix W. During training, we essentially solve an optimization problem minimizing the loss of the given training data. The choice of the optimizer is heavily contributing to the result. All common optimizers work in the same fashion: they begin with some initial parameter estimate $W^{(0)}$ and update this estimate according to some rule $W^{(t+1)} \leftarrow W^{(t)} + \eta^{(t)}\vec{s}^{(t)}$. Their main difference is the choice of the search-direction $\vec{s}$ and the learning-rate (or step-size) $\eta^{(t)}$ for step (t). The parameters are updated until either a maximum number of iterations is reached, or some convergence criterion is fulfilled. After trying different available options, we found the ADAM [73] optimizer to have the best performance in our case.

ADAM is based on the well-known stochastic gradient descent method (SGD)[78], but it uses a concept called *momentum* to further improve the convergence of the optimization. The so-called momentum is conceived by observing a common problem in gradient descent-based optimizers. In cases where the eigenvalues of the Jacobian of the loss function with respect to the weights vary strongly, gradient descent oscillates between slopes while only slowly converging towards the minimum [79]. Using the momentum ($m^{(t)}$) aims to improve the convergence by adding a fraction $\rho_1 \in [0, 1)$ of the previous search direction to the current one:

$$\begin{aligned} m^{(t)} &:= \rho_1 m^{(t-1)} + (1-\rho_1)\frac{dL}{dW} \\ W^{(t+1)} &\leftarrow W^{(t)} - m^{(t)}. \end{aligned} \tag{3.5}$$

Where $m^{(t)}$ is the momentum, L is the loss function and W are weight matrix entries. This speeds up the optimization. In addition to the first momentum in equation (3.5), ADAM uses the second momentum

$$v^{(t)} := \rho_2 v^{(t-1)} + (1-\rho_2)\frac{dL}{dW} \odot \frac{dL}{dW}. \tag{3.6}$$

Where $\odot$ is the entry-wise product. The moments are initially set to be zero-vectors $m^{(0)} = \vec{0},\ v^{(0)} = \vec{0}$ which leads to a bias towards $\vec{0}$, especially during the first few

steps. The parameters $\rho_1, \rho_2 \in [0, 1)$ counteract this issue by exponential decay adjusting of the moments. We use values of $\rho_1 = 0.9$ and $\rho_2 = 0.999$ as suggested by [73].

$$\hat{m}^{(t)} := \frac{1}{1-\rho_1^t} m^{(t)},$$
$$\hat{v}^{(t)} := \frac{1}{1-\rho_2^t} v^{(t)}. \tag{3.7}$$

ρ_1^t is the t-th power of ρ_1 (the same holds for ρ_2^t). The ADAM update rule is then

$$W^{(t)} \leftarrow W^{(t-1)} - \eta \frac{\hat{m}^{(t)}}{\sqrt{\hat{v}^{(t)}} + \epsilon}. \tag{3.8}$$

Kingma et al. [73] suggest a global learning-rate of $\eta = 0.01$ and a smoothing value of $\epsilon = 10^{-8}$. Empirical evidence in [73] shows that ADAM outperforms other optimizers in terms of convergence speed and quality of the found solution in many cases.

Loss function

The most commonly used loss function for regression problems in neural networks is the mean squared error (MSE). Previous results [72] have shown that the **mean square percentage error** (MSPE) is better suited for our specific application. This is because the measured run times have different orders of magnitudes and the mean squared error penalizes deviations on the larger data points much more. The MSPE is given by

$$L^{\text{MSPE}} := \frac{100\%}{m} \sum_{k=1}^{m} \left(\frac{y_k - f(p_k)}{y_k} \right)^2, \tag{3.9}$$

where y_k are the actual measured run times and $f(p_k)$ are the corresponding predictions. Compared to the mean squared error, the MSPE loss is relative to the magnitude of the pointwise data.

Regularization

Due to the high number of weights, neural networks may suffer from over-fitting. Meaning that, after training, the performance is good on the training data set, but

poor on test data due to a lack of generalization. We found out that using the dropout technique [74], which reduces the number of weights, improves the training results significantly. Dropout randomly deactivates some neurons during each training step.

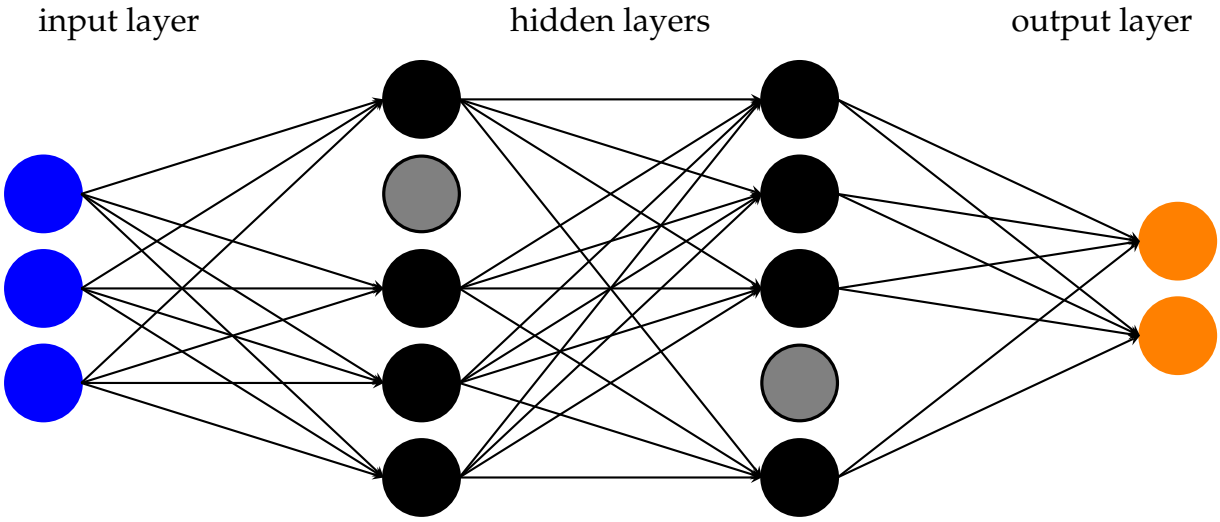


Figure 3.3: A neural network with two hidden layers with dropout applied. Gray circles represent disabled neurons.

Figure 3.2 shows a fully connected neural network with two layers and three neurons for the input, five for hidden layers, and two for the output layer. Applying a dropout of $p = 0.2$ to a five-neuron layer means choosing 1 neuron randomly (uniform distribution) and disabling them, as shown by the gray neurons in Fig. 3.3. With this, only the remaining neurons' weights are updated. In the next training step, a different set of neurons are selected and disabled, again randomly, the previously deactivated neurons are resumed (with the weights that they had before deactivation) and training is applied to all active neurons.

3.3 Computational Resources Distribution

In this section, an optimization problem is derived which uses the performance models to calculate the core distribution with the lowest total run rime. The optimization procedure has been initially described in [67, 68]. The description is modified in this section for a better understanding. We show that the interpretation of the optimization problem directly depends on whether we execute the involved solvers simultaneously or one after the other.

The inter-solver load balancing aims to find the optimal assignment of cores for each solver for a limited number of available cores Q, such that the overall run time $F(q_1, \ldots, q_l)$ is minimized. This can be expressed by the following optimization problem:

$$\begin{aligned} &\underset{p_1,\ldots,p_l}{\text{minimize}} \quad F(p_1, \ldots, p_l) \\ &\text{subject to} \quad \sum_{i=1}^{l} p_i \leq Q. \end{aligned} \tag{3.10}$$

Where p_i is the core count for solver i. The optimization process that we used in the current work is schematically shown in Fig.3.4. The function F depends on the single solver run time f_i and the choice of the iterative coupling scheme, in particular on whether both solvers are executed simultaneously (parallel coupling) or one after the other (serial coupling). The interpretation of this optimization problem depends

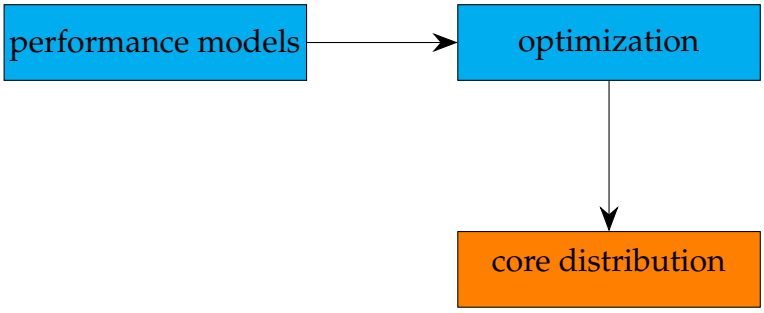
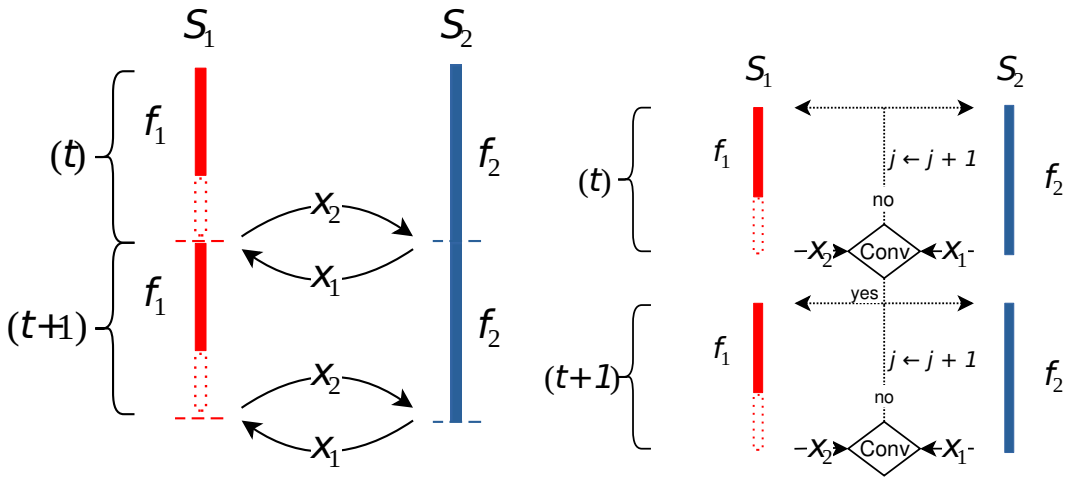


Figure 3.4: Load balancing across solvers: workflow for finding the optimal core assignment.

on the coupling scheme. If a parallel coupling scheme (implicit or explicit) is used, the waiting time of the solvers must be minimized. In other words, the run times that different solvers spend for one iteration or time step must be as similar as possible. Figure 3.5 schematically explains the idea. In this case, the optimization problem can be written in the following form:

$$\begin{aligned} &\underset{p_1,\ldots,p_l}{\text{minimize}} \quad F(p_1, p_2, \ldots, p_l) \quad \text{with} \quad F = \max_i f_i(p_i) \\ &\text{subject to} \quad \sum_{i=1}^{l} p_i \leq Q. \end{aligned} \tag{3.11}$$

If we use a serial coupling scheme (implicit or explicit), the sum of the run time of all solvers must be minimized. This can be understood from Fig. 3.6. In this case,



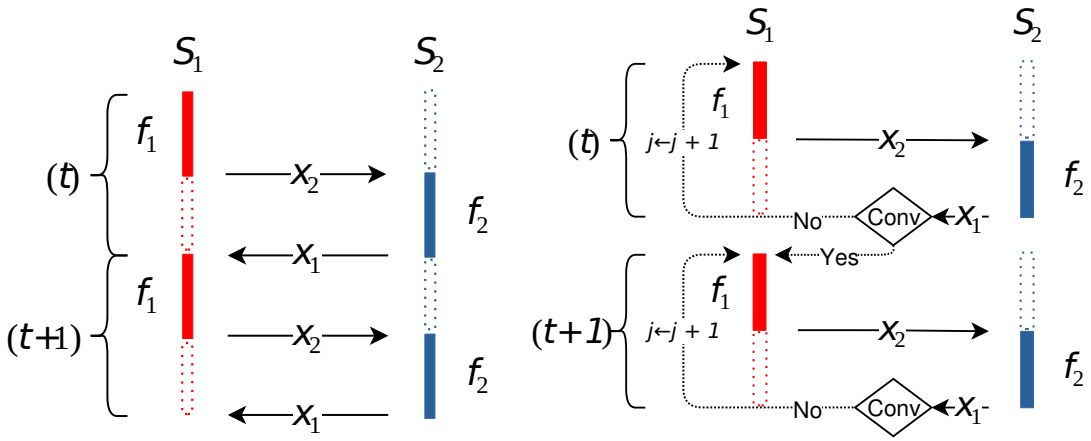
(a) Explicit-parallel scheme. (b) Implicit-parallel scheme.

Figure 3.5: Inter-solver load balancing problem: execution diagrams for explicit and implicit parallel coupling schemes. Solid bars are the time that the solver is running, while, dashed bars indicate that the solver is not performing any calculations and the associated cores are idling [68].

the optimization problem will be turned to the following form:

$$\begin{aligned} &\underset{p_1,\ldots,p_l}{\text{minimize}} \quad F(p_1,\ldots,p_l) \quad \text{with } F = \sum_{i=1}^{l} f_i(p_i) \\ &\text{subject to} \quad \sum_{i=1}^{l} p_i \leq Q. \end{aligned} \tag{3.12}$$

The available cores must be distributed in a way that the total time spent for one iteration is minimized. Note that, following a serial coupling scheme where solvers can not share the available computational resources always imply substantial idle times. Consequently, this approach is less efficient compared to the parallel method and is not followed in the current work. However, since both methods are used in literature (For instance, see [29]), we included the formulation for the sake of completeness. If the f_i are given by the PMNF regression, this optimization is a nonlinear, possibly non-convex integer problem. It can be solved by the use of branch and bound techniques. However, we introduce a new constraint that simplifies the solution procedure. The introduced performance models are able to estimate the scalability limit of each solver, i.e., the point where further increasing the core count will increase the run time and, thus, should be avoided. Using this limit as an upper bound for the core count, we can safely assume that the f_i are monotonically decreasing, i.e., assigning more cores to a solver never increases the run time (within the solvers'



(a) Explicit-serial scheme. (b) Implicit-serial scheme.

Figure 3.6: Inter-solver load balancing problem: execution diagrams for explicit and implicit serial coupling schemes. Solid bars are the time that the solver is running, while dashed bars indicate that the solver is not performing any calculations and the associated cores are idling [68].

scalability limit). Based on this, we can simplify the constraint in Eq.(3.10) to

$$Q = \sum_{i=0}^{l} p_i. \tag{3.13}$$

With this, the problem can be solved by brute-forcing all possible assignments for p_i as Eq.(3.13) drastically limits the amount of combinations we have to try to $\binom{Q-1}{l-1} = \frac{(Q-1)!}{(Q-l)!(l-1)!}$. Since l (number of coupled solvers) is usually small ($l = 2$ or 3 in our test cases), the optimal solution can be quickly calculated.

3.4 Multi-Step Inter-Solver Load Balancing

The load balancing overhead for complex problems can be very high, especially when a large computational grid is used. The performance of the load balancing method strongly depends on the accuracy of the performance models. In case the target computational mesh is very large, running a large number of simulations with the target mesh is extremely expensive. As explained in Sec. 3.2.3, one way to approach this issue is to build multi-variable performance models that can be established with cheaper performance data from simulations that use smaller computational meshes. However, the multi-variable performance modeling requires a fairly high number of training data that are not available for all cases. We propose a

multi-step approach for these cases. The proposed method needs performance data only for two computational meshes and thus, keeps the overhead cost low, while it preserves the accuracy of the models.

We establish the performance model and the core distribution in three subsequent steps: (i) We initially run short simulations using a small mesh to gather initial performance data. These simulations are very cheap due to the small problem size. We use these data to create an initial performance model for each solver; (ii) for these models to be applicable to the target mesh, we scale their output with the ratio between the target mesh size and the small mesh size; (iii) in the next step, we run only a few short simulations with the target mesh using the core distribution obtained from the derived models to obtain more accurate performance data; (iv) we establish new performance models using these new data. This last step is cheaper than the one-step model generation for a comparable accuracy since the core distribution is already close to the final distribution such that the respective regression models are very accurate even for a small number of data points. Note that all the simulations for regression data generation are only performed for two time steps. Thus, their simulation time is very small compared to full FSI simulations.

The test cases that are investigated in this chapter, either have a small mesh or we have access to enough performance data to build multi-variable performance models. Therefore, the multi-step method will not be used in this chapter. Instead, we will use this method for the test case presented in Chapter 5.

3.5 Load balancing with APES

To prove the efficiency of the proposed load balancing method, we compare the results with a solver-specific method developed for the APES [65] framework. This method is chosen since the test cases that will be presented in the Sec. 3.6 will use solvers from this framework. The method's description is taken from [68] and is repeated in this section for the sake of completeness.

"To do load balancing for simulations carried out within the APES framework, the SPartA algorithm [80, 81] is utilized. This algorithm is implemented in the common data structure TreElM [82] of APES and can be employed by all solvers in the framework. The objective of this algorithm is to shift elements from one process to another, to enable the same or similar workload on each process involved in the computation. The SPartA algorithm is based on space-filling curves and weights per

element, providing information about the actual load of each element.

The method is suitable for simulations, in which the configuration of the simulation setup does not change, i.e., no dynamically adaptive meshes are used. Timers are placed in compute-intensive routines, among others in the routines where the physical flux is computed as well as the projection onto the test function for the underlying scheme. In the context of multi-scale and coupled problems, timers are included to capture the workload of elements that are involved in the coupling as well. Furthermore, multilevel meshes as well as the presence of geometries in the simulation domain are captured by the timers [83]. With that, an informed statement about the workload of each element can be made. The `MPI_Wtime` is used to measure the respective times. The load balancing procedure is realized through weight files, that are dumped after a successful simulation onto the disk and used to redistribute the workload among the processes. The simulation is restarted and the SPartA algorithm reads those weight files (each subdomain has one) to calculate the new partitioning for each subdomain [83]."

3.6 Performance Results

To demonstrate the efficiency of the proposed data-driven load balancing scheme, we examine it in two different test cases. The first test case investigates a Gaussian pressure pulse spreading over a cubic domain. The domain is decomposed into an inner and an outer subdomain, solved by using different mesh resolutions and scheme orders. The pulse spreads from the inner domain to the outer part. For this test case, we compare the load balancing results for both the proposed data-driven scheme and the method implemented in APES explained in Sec. 3.5. In addition, we evaluate the effect of applying the proposed method on the simulation run time for various core numbers. Furthermore, we present the initial results of multi-variable performance modeling for this test case using both EPMNF regression(Sec. 3.2.2) and neural networks (Sec. 3.2.3).

The second test case investigates a very complex real-world test case. We evaluate the proposed load balancing method for the simulation of fluid-acoustics interaction around an aircraft wing. The simulation domain is divided into three subdomains. In the innermost domain, the fully compressible Navier-Stokes equations are solved. For the intermediate domain, the simpler Euler equations are solved, while, for the outer domain, only the linearized Euler equations are considered. For this test case,

we present the scalability and the performance measurements and show that the data-driven method can efficiently distribute the computational resources among the subdomains and significantly suppress the load imbalance. The scalability results are presented with and without load balancing to evaluate the significance of effective load distribution.

For both test cases, each subdomain is treated independently by using a separate solver to solve different equations with different mesh resolutions and scheme orders. A coupling library is incorporated for data communication, data mapping, and coupling between the single-physics domains. We use instances of the solver Ateles [84] which is an explicit time-integrating solver which is integrated into APES and uses the modal Discontinuous Galerkin Method (DGM) for discretization. The parallel and efficient fluid solver is capable of solving different equations, including the compressible Navier-Stokes, the inviscid Euler, and the linearized Euler equations. Ateles is specifically geared towards high-order schemes and Cartesian elements. The restriction of the numerical scheme to Cartesian elements allows a dimension by dimension strategy for various operations involving the basis functions in elements [65]. To couple single-physics subdomains, we use the preCICE coupling library which provides (1) the communication between different solvers, (2) data mapping for the coupling variables, and (3) iterative equation coupling for surface coupled problems in a modular way [34]. To use preCICE, each solver needs an independent `MPI_COMM_WORLD` and inter-solver data communication channels are established by using `MPI ports` or lower-level TCP/IP sockets. In the current work, lower level TCP/IP sockets are used. For data mapping between non-matching meshes, a nearest projection data interpolation method is used. For the investigations with the method used in APES (only for the comparison in the first test case), we consider an internal coupling approach. Here, a single `MPI_COMM_WORLD` can be used for all subdomains. The test case description and initial performance analysis for both test cases are presented in [67, 68]. This chapter extends the investigations for multi-parameter performance modeling and provides more detailed analysis on the effect of the proposed load balancing on the simulation run time.

3.6.1 Hardware Description

The scalability measurements are carried out on the SuperMUC-NG supercomputer at the Leibniz Supercomputing Centre of the Bavarian Academy of Science and Humanities (LRZ). This machine consists of 3.1GHz Intel Xeon Platinum 8174 (Sky-Lake) processors. Each node contains two processors with 24 cores per processor

(48 cores per node) and 96GB of RAM. The nodes are connected via Intel Omni-Path interconnect.

3.6.2 Gaussian Pressure Pulse Test Case

In the first step, we consider a simpler test case to study the effect of the proposed load balancing method. This test case investigates a Gaussian pressure pulse traveling in a cubic domain. To reduce the simulation complexity, the domain is decomposed into an inner and an outer subdomain coupled using a parallel explicit scheme. Only in the inner domain, a fine mesh is used, while the outer domain uses a coarser mesh. This reduces the simulation complexity, by decreasing total degrees of freedom, while preserving the simulation accuracy. Initially, a Gaussian pressure pulse is located in the inner domain which spreads with the speed of sound to the outer domain. This test case is simple but representative, since each subdomain solves a different set of equations, and has a different scheme order and a different spatial discretization. Hence, each subdomain has a different workload and therefore load imbalance can be foreseen, which needs to be addressed appropriately.

The complete simulation domain is a $5 \times 5 \times 5$ unit length cube, which is decomposed into an inner subdomain of size $1 \times 1 \times 1$ and an outer subdomain (see Fig. 3.7). Initially, the pressure and density values are 10^5 and 1.0 respectively. The inner subdomain is solved using the inviscid compressible Euler equations, while for the outer subdomain, the linearized Euler equations are used. The inner subdomain is solved with a much finer mesh, comprising 262,144 elements, while the outer domain has a much coarser mesh, comprising 507,904 elements [85]. The mesh resolution and scheme order for the inner and outer subdomain are provided in Table. 3.1. This setup results in 221,184 coupling points on the common boundaries for the inner and 55,296 for the outer subdomain (non-matching mesh on the interface). The initial perturbation of all state variables (density, momentum, and energy) are set to zero in the outer domain, while the background density and pressure are same as in the inner domain. The time step size is fixed for both subdomains and has a value of 10^{-6} time units.

Load Balancing and Scalability Analysis

We evaluate the proposed load balancing method by comparing the load imbalance between the inner and the outer domain for the proposed data-driven method and

Table 3.1: Gaussian pressure pulse test case for the inter-solver load balancing: mesh resolution and scheme order for inner and outer subdomains for performance analysis [68].

	Euler inner domain	Linearized Euler outer domain
Domain [x, y, z]	[1, 1, 1]	[5, 5, 5]
Elements	262,144	507,904
Scheme order	3	6
nDof	283,115,520	871,055,360

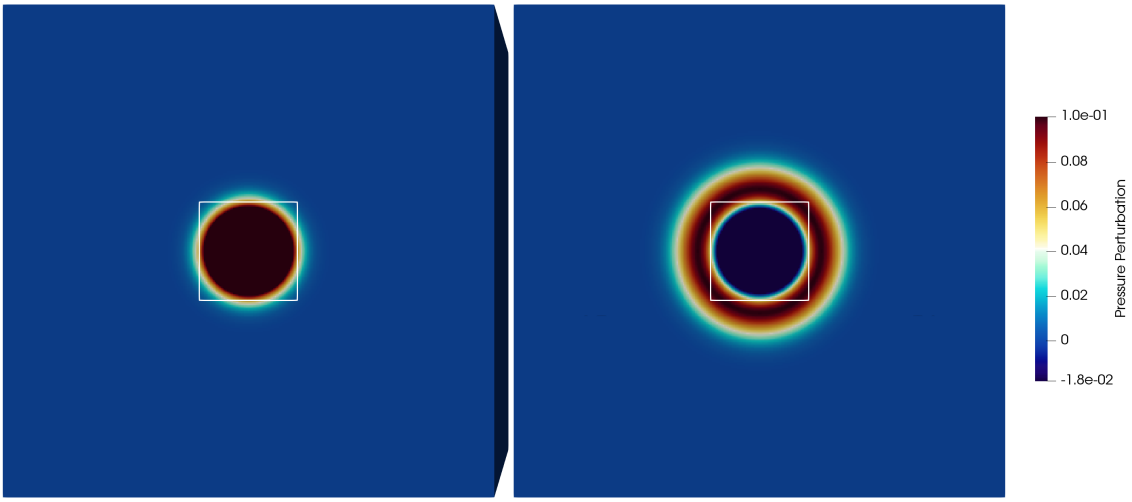
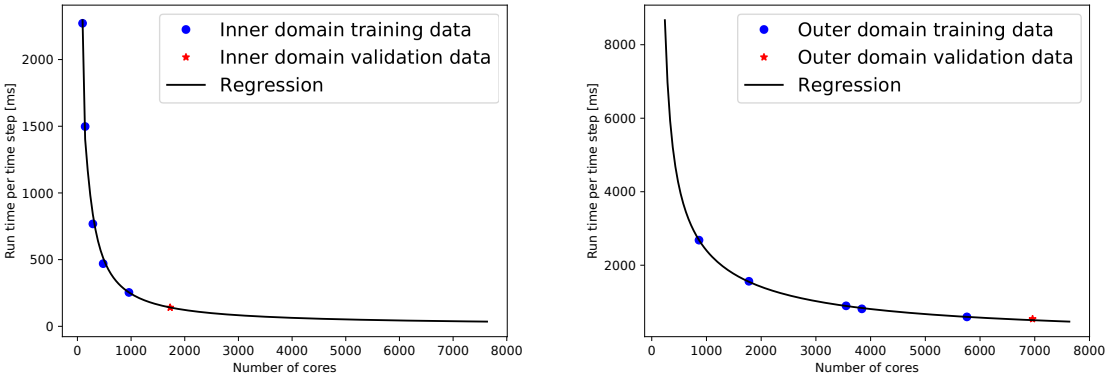


Figure 3.7: Gaussian pressure pulse test case for inter-solver load balancing: The pressure pulse spreads over time from the inner subdomain to the outer. The white frame highlights the coupling interface. This illustration is taken from [68].

the method integrated into APES. For the data-driven method, we need the performance models for each subdomain. We run 6 coupled simulations (5 for training and 1 for validation) with different core counts and incorporate the PMNF regression to find appropriate models for each solver. These models for the inner and the outer domain are depicted in Fig. 3.8. The validation data point (red dot) shows that the PMNF regression is able to accurately model both solvers' performance.

In addition, the overall scalability of the current test case is presented in Fig. 3.9 which shows that the solver scales up to about 12,000 CPU cores for this particular coupled configuration. The scalability results are presented also without any load balancing, where the cores are distributed among solvers proportional to their mesh size (number of degrees of freedom). The comparison shows that applying the data-

driven load balancing method results in about 25% run time reduction for the higher number of cores. Furthermore, Fig. 3.10 compares the load imbalance between the



(a) Inner domain. (b) Outer domain.

Figure 3.8: Gaussian pressure pulse test case for inter-solver load balancing: Data driven performance models for the inner and the outer domains. The method suggests $f_{inner}(p) = 2.4e4p^{-0.5}log(p)^{-2} - 5.8e4p^{-2}log(p)^{-1}$ and $f_{outer}(p) = 4.8e3p^{-1.25}log(p)^{2} + 1.3e5p^{-1.5}log(p)^{-1}$ as performance models for the inner and the outer domain, where p is the number of cores.

subdomains (inner and outer) using both methods for different core numbers. The load imbalance is calculated as follows:

$$load\ imbalance = \frac{f_{outer} - f_{inner}}{f_{outer}} \times 100$$

where f_{outer} and f_{inner} are average run times (per iterations per core) for the outer and the inner subdomain, respectively. The figure shows that the proposed method is able to successfully remove the load imbalance between the domains and, in most cases, the load imbalance is about 1%. For a smaller core count, we can observe a higher run time difference between the inner and the outer domain. The reason is due to the system used for the computation (SupermucNG), which does not allow the sharing of a single node between the different subdomains, as they are executed separately. As mentioned earlier, for coupling with preCICE, each solver needs an independent `MPI_COMM_WORLD`. Hence, each subdomain is computed simultaneously but executed independently. Therefore, we always have to use complete nodes (48 cores) for each subdomain, which, however, prevents the optimal distribution of the available computational resources. For coupling with the internal APES library, we do not face this constraint, since a single `MPI_COMM_WORLD` can be used for all subdomains. For larger core numbers, the situation is better since the number of available nodes is large enough for a fine distribution. We must emphasize that this

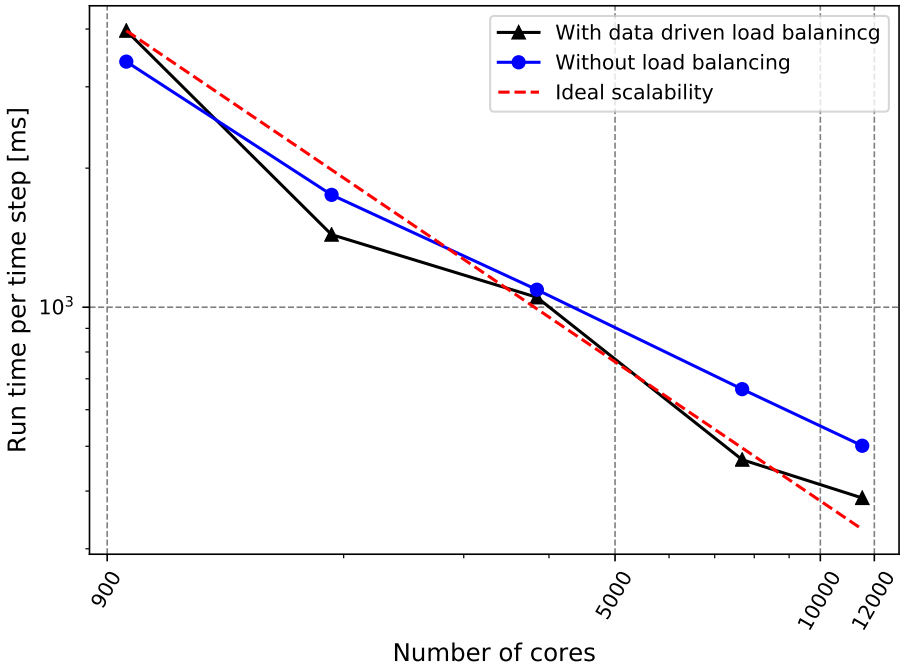


Figure 3.9: Gaussian pressure pulse test case for inter-solver load balancing: Average run time per time step for different core counts (strong scalability measurement). Results are presented with and without load balancing.

is a limitation of the hardware, not the proposed method. Therefore, on a computing system that allows node sharing between jobs, we expect a better load balancing for smaller core numbers.

In the case of the load balancing with APES, we can observe mostly less than 1% remaining load imbalance. In the case of the highest core count, the load imbalance increases. This seems to be due to the perfect linear scalability assumption which is not true for a higher number of cores. The data-driven method shows better performance for higher core numbers since the data-driven performance models are able to predict the solver's run time accurately based on the performance data gathered from sample simulations which reflect the actual behavior of each solver.

Furthermore, to evaluate the overall effect of the load balancing method on computational efficiency, we analyze the associated overhead. The main part of the load balancing overhead comes from the small simulations that are carried out to build the performance model functions. For the current test case, we used 6 small simulations with an accumulated total run time of 29.4 seconds. After the performance models are established, the extra cost to evaluate the core distribution for each simulation is insignificant. Therefore, this overhead is only paid once, and not repeated

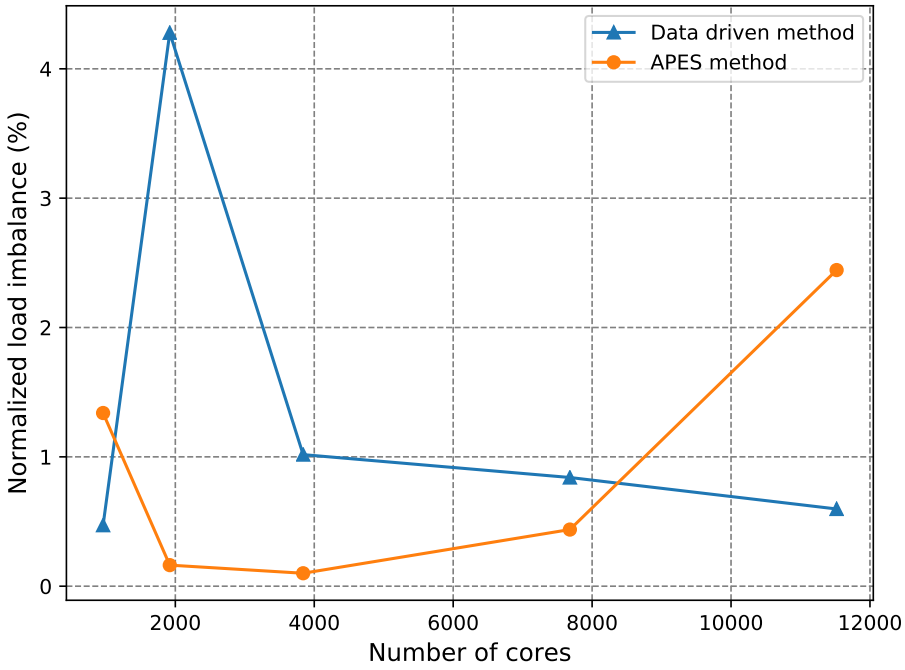


Figure 3.10: Gaussian pressure pulse test case for inter-solver load balancing: Comparison of the load imbalance between inner and outer domains for the data driven and the method integrated in APES. This illustration is taken from [68].

for each simulation. This overhead for load balancing is relatively small, compared to the consequent reduction in the computational cost. For instance, the reduction in the run time due to the load balancing method on 11,520 cores is around 23%, which means the reduction for a complete simulation with 2000 time steps accumulates to 241.9 seconds. The overhead is only about 12% of the saved time and thus, it is very well justified [68].

Multi-Variable Performance Modeling

Up to now, we have only presented the performance modeling for only one variable, the core number. In this section, we present the preliminary results for a multi-variable performance model. In addition to the core number, we include the problem size (number of elements) and present the result for the same Gaussian pulse test case. Since we only present the performance modeling in this section, the results are presented only for the inner domain. The test case setup is similar to the Sec. 3.6.2. In addition, the used mesh resolutions are provided in Table. 3.2. For both the EPMNF model and the neural network model, training data are measured performance data

Table 3.2: Gaussian pressure pulse test case for inter-solver load balancing: various mesh resolutions for multi-variable performance modeling of the inner domain.

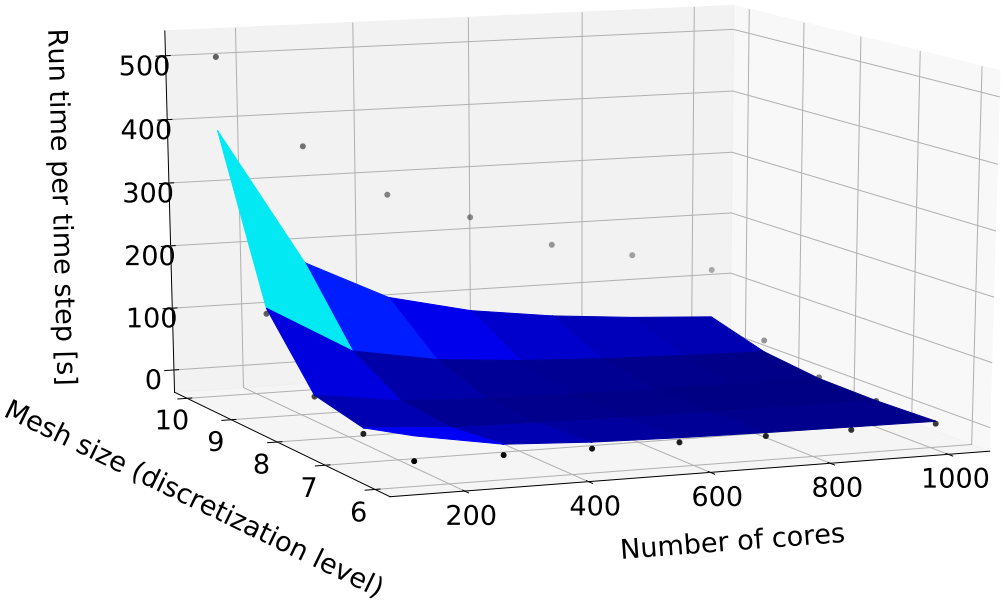
Discretization level	6	7	8	9	10
No. of elements	4096	32,768	262,144	2,097,152	16,777,216

for the mesh resolutions (provided in Table. 3.2) for the various number of cores (for 144, 288, 432, 576, 720, 864, and 1008 CPU cores). The performance model for EPMNF regression is shown in Fig. 3.11. For performance modeling, the hierarchical search approach is used to find the best-combined base functions in multi-variable modeling. Similar to the PMNF models, only two terms are used for the regression.

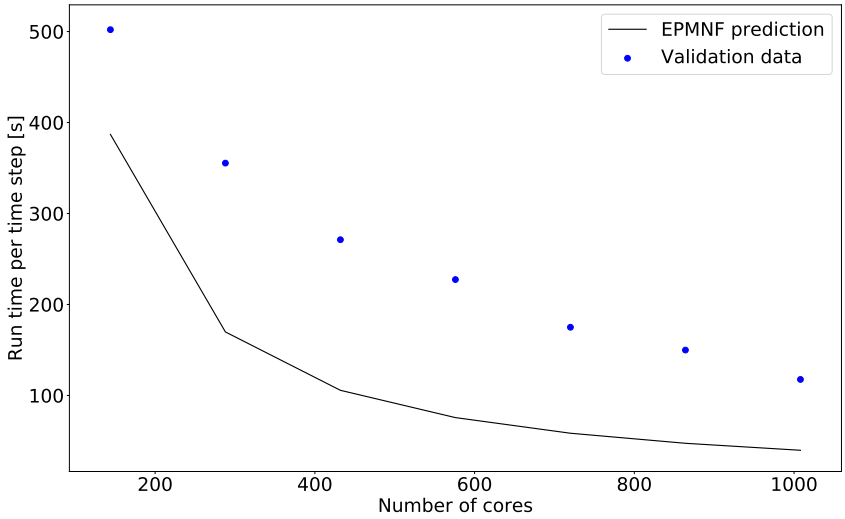
From Fig. 3.11a, it seems that the regression is able to model the run time against both core number and mesh size, but looking closer at the test data (e.g. Fig. 3.11b) shows that the predicted run time is not accurate at all. It seems that the model is not even able to learn the run time of the training data. This is probably because using only two terms for the regression is not enough. Using more terms is not easily feasible in our simple approach, because increasing the number of terms will increase the search space, and finding the base functions can be excessively expensive. It should be repeated here, that the EPMNF regression is introduced to model a single kernel's performance and not a solver which consists of many complicated kernels. Each kernel can have a different behavior which makes the behavior of the solver complicated enough that it can not be modeled by using only two regression terms. Therefore, we take an alternative approach and use a neural network for the performance modeling.

As a first attempt, we use a simple densely connected network with MSPE loss. The used network consists of 3 layers and each layer has 10 neurons. This is the smallest network of this type that can predict the performance accurately. Other types of networks, such as convolutional ones, may need less capacity, but this is still a work in progress. In addition, we have used the ReLU activation function and dropout for training regularization. The optimization method we have used is ADAM, to calculate gradients, a simple back-propagation method is utilized. The neural network performance model is shown in Fig. 3.12. We observed that by using log_2 of the run times for training, the learning and prediction accuracy increased significantly. It is probably because, by applying this mapping, the input and output data have a simpler relationship which makes it easier for the neural network to learn it.

As mentioned, the multi-variable performance prediction results are preliminary at



(a) Performance modeling.

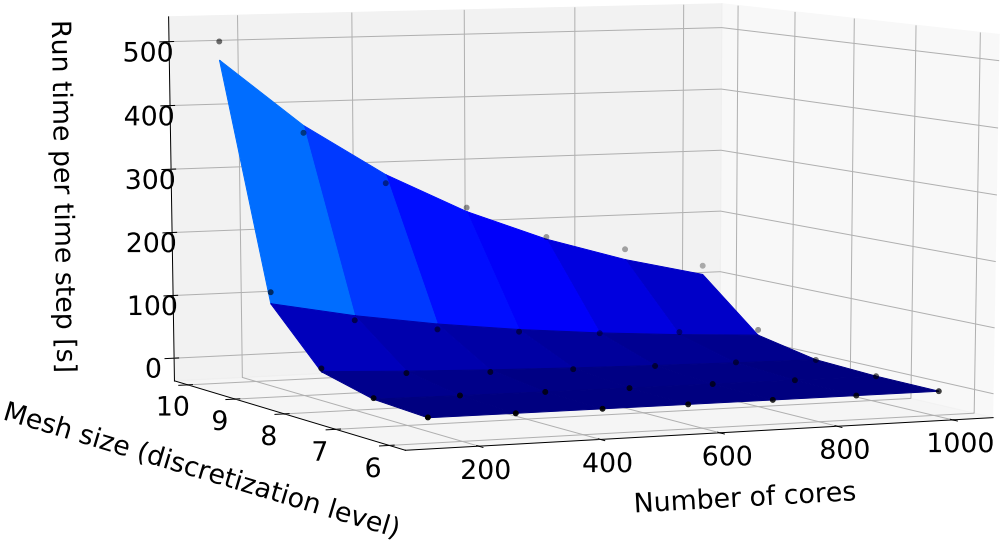


(b) Predicted run time for the mesh with discretization level 10.

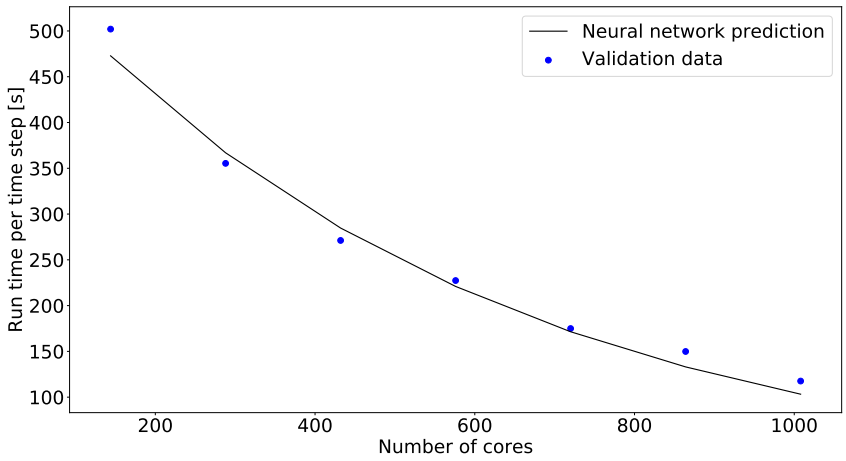
Figure 3.11: Gaussian pressure pulse test case for inter-solver load balancing: Multi-variable (core count and problem size) performance modeling by using the EPMNF regression. The method suggests $f(p,s) = -9.65e3p^{-1}s^{-1}log(p)^{2.5}log(s)^2 + 1.02e4p^{-1}s^{-1}log(p)^3log(s)^1$ as a performance model, where p is the number of cores and s is the mesh discretization level.

this stage. Our approach can be used to provide cheaper performance models for large-scale problems by using the measurements of coarser meshes without losing

the accuracy.



(a) Performance modeling.



(b) Predicted run time for the mesh with discretization level 10.

Figure 3.12: Gaussian pressure pulse test case for inter-solver load balancing: Performance modeling using the neural networks.

3.6.3 Airfoil Test Case

In the second test case, we study the effect of the proposed load balancing on the fluid-structure interaction around an aircraft wing. This is a common phenomenon in many engineering applications, where, due to the presence of a structure and the disturbance of the fluid flow, noise is generated. The problem domain is divided

into three fields, which are solved with different configurations which are coupled following a parallel explicit strategy. This allows for a tailored configuration to capture the occurring physics appropriately with a much lower cost compared to the case in which the whole domain is discretized with a fine mesh. The decomposition of the domain into three sub-domains and the fluid velocity distribution around the airfoil at $t = 1$ second is depicted in Fig. 3.13.

In the innermost subdomain, an airfoil profile (S834) is located, that is modeled as a porous material and is embedded into the equations to be solved (Immersed Boundary Method) [86]. This subdomain is solved by the full compressible Navier-Stokes equations, with a fine mesh resolution and a low scheme order, to capture small scales. Far away from the viscous effects, the fluid dynamics equations to be solved can be simplified to the compressible inviscid Euler equations (middle domain). Here, a coarser mesh resolution and a higher scheme order are applied. In the outermost subdomain, we can further simplify the equations to be solved, as only acoustic wave propagation is expected. Therefore, the linearized Euler equations are solved, where an even coarser mesh and a higher scheme order are used. With this configuration, we make sure to utilize the inherent properties of low dissipation and dispersion error of the high order discontinuous Galerkin scheme for the transport of information over large distances such as the acoustic waves in the far-field. All subdomains are solved with the solver Ateles, while the subdomains are coupled using the library preCICE. The strategy of decomposition allows us to enable complex simulations while reducing the computational effort and considering expensive configurations only where it is necessary and enables simplifications where the physics allows [87, 83].

The configuration of the coupled simulation is provided in Tab. 3.3. The smallest volume is covered by the innermost subdomain, where the expensive compressible Navier-Stokes equations are solved. However, this domain contains the largest amount of elements. This is required to capture small scales and resolve the boundary layer at the airfoil interface. Further, the outermost subdomain, where the linearized Euler equations are solved, has the least number of elements, while considering a scheme order of 9, which is the highest for this coupled scenario. In Tab. 3.3, all length scales are normalized by the chord length of the airfoil (l_c). At the inlet of the Navier-Stokes domain (innermost), a jet is prescribed, that injects a direct stream on the structure. It is located in the exact middle with a radius $r = 0.5$ unit length. The kinematic viscosity μ is predefined with a value of $1.49 \cdot 10^{-5}$. The dimensionless Mach number has a value of $Ma = 0.35$ and the Reynolds number is equal to

Table 3.3: Airfoil test case for inter-solver load balancing: Configuration for the 3-field coupled simulation [68].

	Navier-Stokes inner domain	Euler middle domain	Linearized Euler outer domain
Domain [x, y, z]	[4, 2, 2]	[12, 6, 2]	[12, 3, 2]
Elements	12672	8192	1152
Scheme order	4	6	9
nDoF	4,055,040	8,847,360	4,199,040

$Re = 67,144$. The fluid is streamed into the domain with a velocity, that is ramped up linearly over time and has a final value of $\vec{v} = [0.35 \cdot \sqrt{(\gamma \cdot p/\rho)}, 0.0, 0.0]$, with the isothermal coefficient γ equal to 1.4, the pressure p being 101325 and density $\rho = 1.0$. The perturbation in the outermost subdomain is set to 0.0 initially for all state variables (density, velocity, and pressure) [68].

In total, this coupled scenario has $17,101,440$ degrees of freedom (nDoF). The monolithic approach, where the entire domain is solved with the same equations and scheme order (Navier-Stokes equations and scheme order 4), requires nine times more degrees of freedom [83]. Thus, the partitioned method not only is computationally cheaper but also requires significantly less memory, while it preserves the solution accuracy. For physical results for this test case, please consult [88].

Load Balancing and Scalability Analysis

For this test case, we present the load balancing results for the data-driven method. Similar to the previous test case, we initially need to model the performance of each subdomain. We run 6 coupled simulations (5 for training and 1 for validation) with different core counts and incorporate the PMNF regression to find an appropriate model for each solver. These models for the innermost, the middle, and the outer subdomains are depicted in Fig. 3.14. In addition, the overall strong scalability of the current test case is presented in Fig. 3.15 for the cases with and without data-driven load balancing. For the latter case, the computational resources are distributed according to the problem size (degrees of freedom) among the solvers. The comparison shows that applying the data-driven load balancing method reduces the total run time by about 20% uniformly for all core counts. The scalability limit appears to be imposed by the outermost domain whose parallel efficiency drops after around 4500 cores (total core count). This also appears to be due to the low arithmetic density

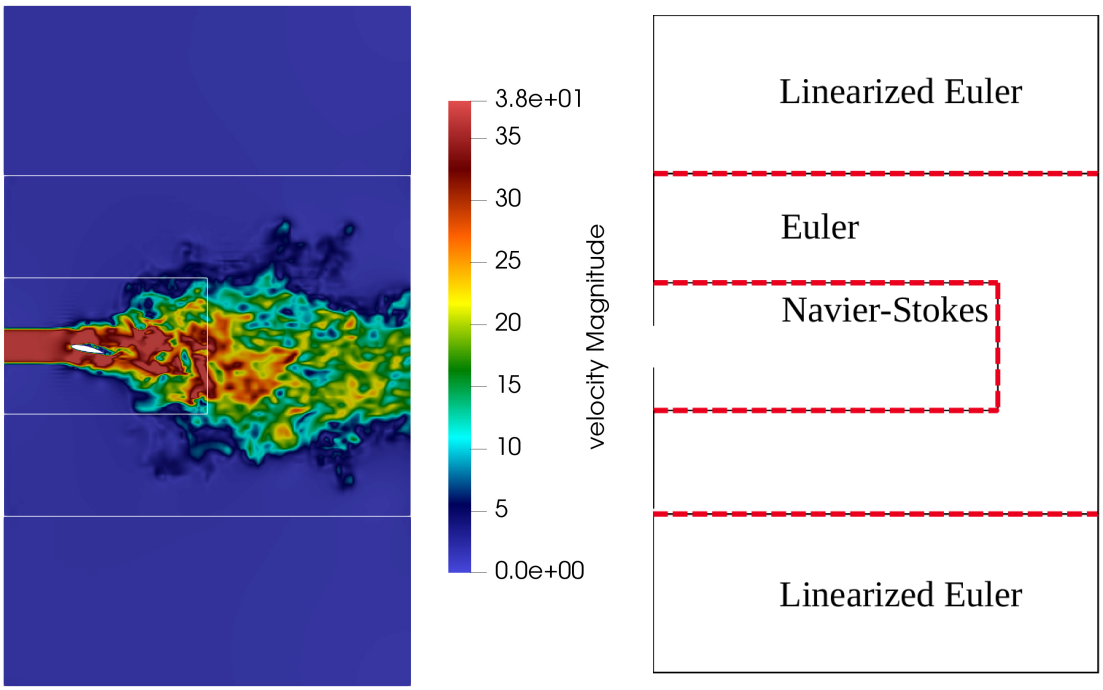
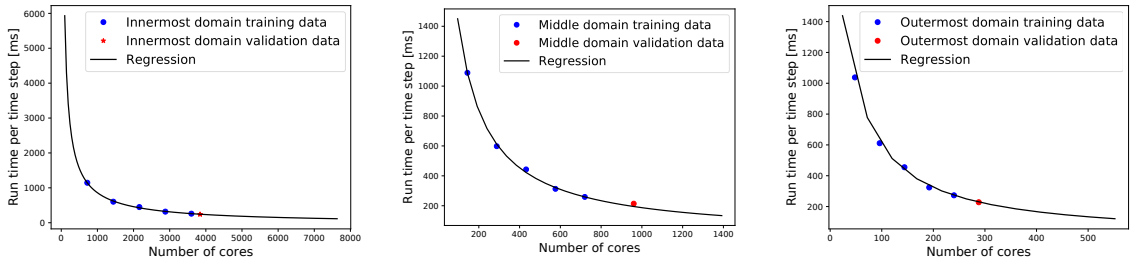


Figure 3.13: Airfoil test case for inter-solver load balancing: (a) Fluid velocity distribution around the airfoil at $t = 1s$. White frames highlight the coupling interface [83]. (b) Decomposition of the simulation domain into three subdomains, solved with the full compressible Navier-Stokes equations (innermost), compressible inviscid Euler (middle) and the linearized Euler equations (outermost). This illustration is taken from [68].

within each core due to the small mesh partition. Fig. 3.16 shows the load imbalance between the subdomains by using the data driven method. The load imbalance is calculated as follows:

$$load\ imbalance = \frac{f_{max} - f_{min}}{f_{max}} \times 100$$

where f_{max} and f_{min} are the maximum and minimum values for the average run time (per number of cores per iteration) of the three subdomains. The figure shows that the proposed method is able to almost remove the load imbalance between domains. Again, we can observe, that for the smallest number of cores (the first point in Fig.3.16), the load imbalance is the highest. In this test case, different subdomains have very different workloads. The most compute-intensive domain is the innermost subdomain, where the Navier-Stokes equations are solved. While the outermost subdomain needs only a few cores for the computation as its physical complexity is much lower when compared to the innermost domain. This is also the case for



(a) Innermost domain. (b) Middle domain. (c) Outermost domain.

Figure 3.14: Airfoil test case for inter-solver load balancing: data driven performance models for the innermost, middle and the outermost subdomain. The method suggests $f_{inner}(p) = -1.9e2p^{-0.5}log(p)^1 + 1.0e4p^{-0.75}log(p)^0$, $f_{middle}(p) = 2.0e3p^{-1.25}log(p)^0 + 1.4e3p^{-1.25}log(p)^1$ and $f_{outer}(p) = 1.0e3p^{-1.25}log(p)^0 + 4.3e2p^{-1.5}log(p)^1$ as performance models for the innermost, the middle and the outermost domain, where p is the number of cores.

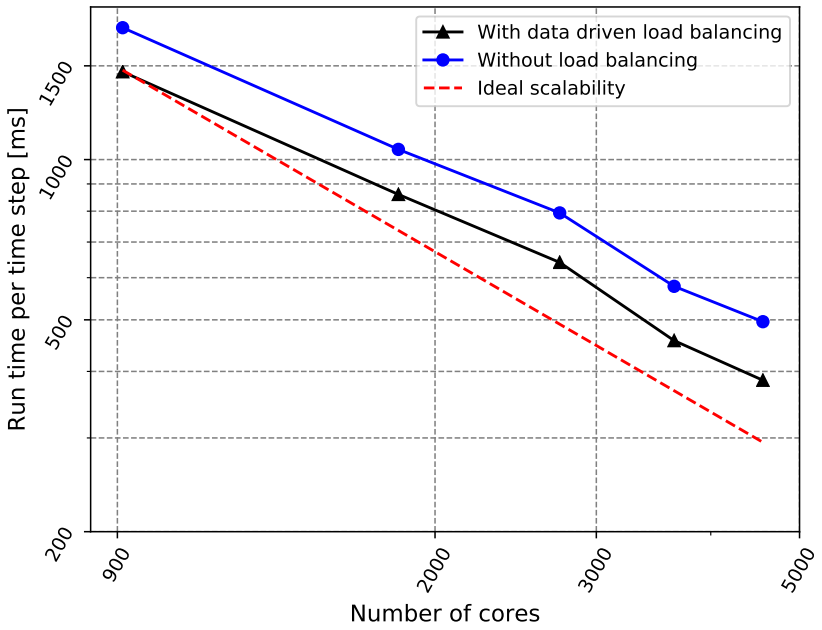


Figure 3.15: Airfoil test case for inter-solver load balancing: Average run time per time step for different core counts (strong scalability measurement). Results are presented with and without load balancing.

the middle domain, which is less expensive than the innermost, with roughly 3.5 times faster computation. Considering all these differences in the complexity of the subdomains, it is apparent that only a fine distribution of available cores allows for overcoming the load imbalance. However, we can assign cores to subdomains at the

level of whole nodes only, due to hardware limitations, leading to idling and waiting for the other two subdomains in order to exchange data. With increasing number of cores, we observe that the load imbalance reduces.

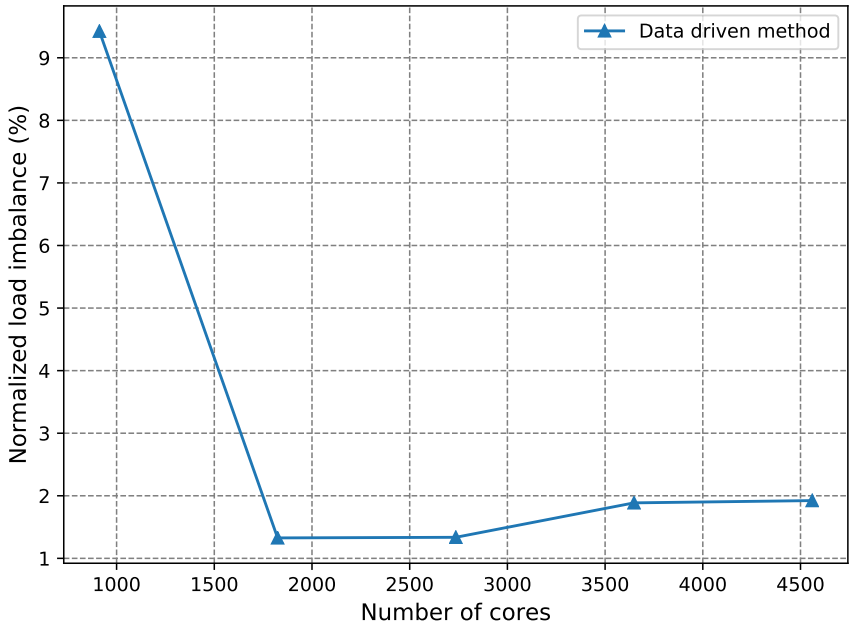


Figure 3.16: Airfoil test case for inter-solver load balancing: Load imbalance between domains using the regression based data driven method. This illustration is taken from [68].

Similar to the previous test case, we analyze the proposed data-driven load balancing overhead for this test case to evaluate the effect of the load balancing method on the computational efficiency. In total, 6 small simulations were carried out to produce the required data for establishing performance models. The accumulated run time for these simulations is 32.6 seconds. The extra cost to evaluate the core distribution for each simulation is very small in this test case as well. For the current test case, we need to simulate at least 0.5 seconds of physical time to achieve useful results. The appropriate time step size for this problem is 10^{-6} seconds which results in $500,000$ time steps. Therefore, the total overhead for load balancing is negligible, compared to the reduction in the computational cost. For instance, the reduction in run time due to the load balancing method on 4,608 cores is around 11 seconds for one hundred time steps. This means the load balancing overhead is compensated within the first 300 time steps due to the run time reduction. Given the required $500,000$ time steps, the total saved time due to applying the proposed load balancing method is around 15 hours for a complete simulation [68].

3.7 Summary

A data-driven performance modeling and load balancing method for multi-physics simulation are introduced in this chapter. To efficiently distribute cores among solvers, we first used the PMNF regression to model the performance of each solver against the number of cores. Then, an appropriate optimization problem is derived and solved to find the optimal distribution. We show two different test cases to evaluate the effect of the proposed method. For both test cases, we use the Ateles solver from the APES framework. Furthermore, the preCICE library is used to couple the subdomains of the partitioned solver. In the first test case, the method is applied to the partitioned simulation of a Gaussian pressure pulse spreading within a cubic domain. The domain is decomposed into an inner and an outer subdomain and each is solved with a different configuration. In the second example, a more complex test case is investigated. We examined the load balancing for fluid-structure interaction phenomena around an aircraft's wing. The simulation domain is split into three subdomains and each is solved with a different configuration. The performance analysis shows that the proposed load balancing method can significantly decrease the run time (up to 25%) and improves the scalability of the simulation. In addition, we showed that the proposed method is able to reduce the load imbalance significantly to around 1%. The results are comparable, even better for higher core numbers, with a solver-specific method developed specifically for the APES framework. However, since we use the run time average (per number of cores per iteration) for performance modeling, this small load imbalance can not be suppressed completely. A possible remedy is using a dynamic load balancing approach to update the core distribution over the run time, but such an approach can introduce other complexities and is subject to future works.

In addition, two different methods for multi-variable performance modeling of the solvers are examined to study the effect of multiple variables on the solver's run time. We used the core number and the problem size as variables. In the first method, the extended version of PMNF regression, called EPMNF, is used for the multi-variable modeling of a solver. The numerical results show that EPMNF with hierarchical search is not able to accurately model the solver's performance with a tolerable cost. Therefore, an alternative approach, which is using neural networks, is taken to study multiple variable cases. The initial results show that neural networks are capable of accurately modeling the performance. They can be used to study the effect of various parameters such as core count, problem size, hardware properties, etc., on the run

time simultaneously. In this work, a simple densely connected network with a ReLU activation function is used. This approach can be used to establish performance models using coarser meshes when establishing the performance models with the fine target mesh is expensive. This work is also a topic for future research.

4 Machine Learning for Convergence Acceleration of Coupled Problems

This chapter presents a data-integrated environment for performance improvement of partitioned FSI simulations. We combine classical physical solvers with a deep neural network (DNN) to accelerate the solution convergence. The DNN provides an estimation for each time step of the simulation. The estimated solution is used by classical solvers as an initial guess to compute the final solution. The method benefits from both the partitioned and the monolithic approach. This is achieved by decomposing the simulation domain into a solid and a fluid subdomain to use available solvers for each (partitioned approach advantageous) and using only a single neural network for the entire domain (no iterative procedure needed, benefit of the monolithic approach). The DNN is pre-trained based on data generated over several time steps of the classical simulation. To ensure that the DNN produces suitable results in all time steps of long simulations, we introduce an on-the-fly re-training scheme to keep the network updated and maintain its prediction accuracy by avoiding the so-called model-data inconsistency during the simulation. In addition, we propose a parallelization scheme for the training process of the neural network which is compatible with the techniques used in the classical solvers. This is a crucial step for integrating deep neural networks with highly parallel physical solvers to investigate large complex problems.

We provide numerical evidence to show, that the proposed method significantly improves the performance of the FSI simulation. We present a numerical convergence analysis for a one-dimensional simulation of flow in a deformable tube. The numerical investigations show, that the neural network acceleration scheme with the on-the-fly re-training is able to significantly reduce the number of FSI solution iterations. Furthermore, our initial investigation of the proposed parallelization scheme shows, that the training can scale perfectly to 64 CPU cores.

The rest of this chapter is organized as follows. Section 4.1 introduces the application of machine learning methods in simulation science and explains the approach that we address in this chapter in more detail. Sec. 4.2 presents the data-integrated scheme for FSI simulations. This includes the coupled solver design, the neural network architecture, and the on-the-fly training strategy. Section. 4.3 explains the

parallelization method for the neural network training. The numerical investigations to prove the effectiveness of the proposed methods are presented in Sec. 4.4, while Sec. 4.5 summarizes and concludes the chapter.

4.1 Introduction to the Application of Machine Learning in Simulation Science

Machine learning methods have shown promising performance in various scientific fields. One potential usage of these methods is their application in improving the performance of numerical simulations, i.e., data-integrated simulation. Given the vast amount of simulation and experimental data, enough data are available for training data-driven models. These models can be either integrated into the simulation or used as a stand-alone post-processing tool, e.g., for optimization.

Machine learning, in general, and deep neural networks, specifically, have been vastly investigated for simulation science. For instance, Raissi et al. investigated vortex-induced vibrations of bluff bodies to approximate lift and drag forces by employing deep neural networks [89]. A breakthrough in this field has been achieved by introducing physics-informed neural networks [90, 91, 92]. This class of neural networks integrates the prior knowledge into the learning mechanism by adding the basic physical laws into the loss function. This penalizes the input-output mappings that do not satisfy basic physics and, thus, improves the learning result. Deep neural networks have also been applied to FSI problems. For instance, Gaymann et al. applied DNNs for FSI topology optimization [93]. In the current study, we aim to use deep neural networks to improve the performance of partitioned FSI simulations. We integrate DNNs with classical fluid and solid solvers for an efficient data-integrated simulation.

In addition, we propose a novel parallelization scheme for the training of deep neural networks that is compatible with the parallelization techniques employed in classical simulation software. This is important from various perspectives: (i) Given the size of the training data set in a typical simulation application, efficient training and inference must be considered; (ii) modern classical solvers for investigating complex problems are usually highly parallel. Integrating DNN with these solvers is only possible when the DNN follows a similar parallelization scheme for training and inference.

There are various approaches in the literature to improve the efficiency of the train-

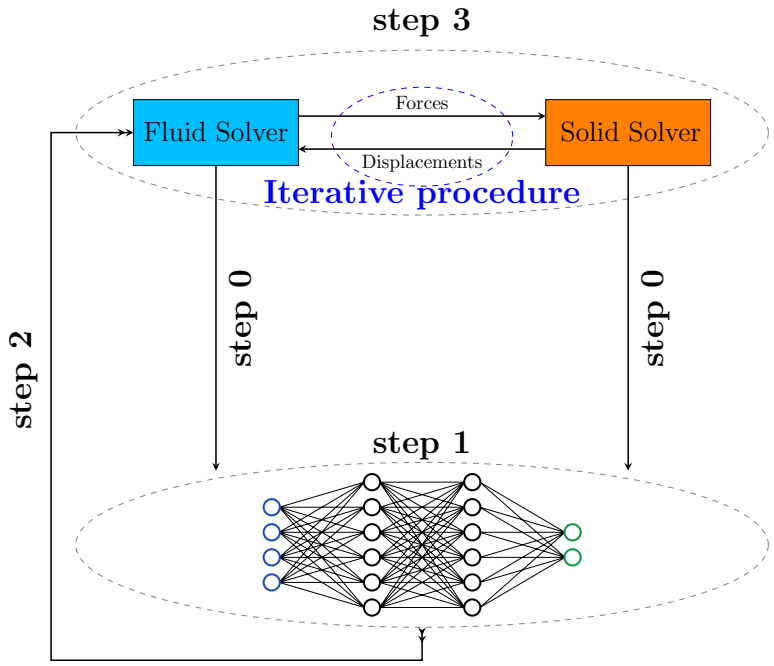
ing, i.e., to reduce the time-to-train, while preserving the learning quality. We can generally categorize these schemes into data parallelization and model parallelization [94]. While the former distributes the training data set among different processes, the latter shares all data among processes but distributes the computations. Both methods need data communication for synchronization. As an example, Vivian et al. [95] presented a data-parallel approach, where the available training data are split into smaller chunks. Each chunk is given to a separate network. Through a global reduction operation, the networks resulting from different training data chunks can share their weights. The weights are averaged and constitute a new network, which is shared among all individual MPI ranks. This procedure is repeated until all available data are fed into the network and the training is completed. This approach is able to reduce the training time. However, it alters the learning algorithm resulting in decreased learning efficiency. In addition, the global reduction operations are potential performance bottlenecks.

The proposed method in the current study is based on decomposing the individual training data sets into smaller spatial sections. This method is primarily targeting simulations in different scientific areas but can be generalized to be utilized for other fields as well.

4.2 Data-Integrated Fluid-Structure Interaction Simulation with Deep Neural Networks

To accelerate the equation coupling between fluid and structure domain in a strongly coupled FSI simulation, we train a neural network to predict the converged solution at the coupling interface at the end of a new time step. The classical solvers use the estimation as an informed initial guess for the implicit coupling iterations (see Sec. 1.1) to calculate the final numerical solution by performing an iterative procedure to solve the fixed point equation. Since the estimated solution is produced by a DNN trained with converged values at the common interface, the classical solvers require fewer iterations for the solution until convergence.

As schematically shown in Fig. 4.1, we do not use two neural networks for the fluid and the structure solver, respectively, but only a single neural network to output the converged solution of a whole coupled time step. This has the advantage that we avoid having to iterate between a fluid and a solid neural network surrogate to get the predicted solution. In addition, we reduce the output to interface data (instead



Step 0: fluid solver transfers common interface pressure at t^{n-1} to NN solver
Step 0: solid solver transfers common interface displacement at t^{n-1} to NN solver
Step 1: NN solver makes a solution prediction for time step t^n
Step 2: NN solver transfers the common interface displacement for time step t^n to fluid solver
Step 3: classical iteartive solution procedure with several iteartions per time step

Figure 4.1: Convergence acceleration in partitioned FSI simulation using surrogate neural networks: schematic view of using neural networks to accelerate the partitioned simulation of fluid-structure interaction problems. A neural network is trained to receive simulation results at time step t^{n-1} (step 0) and compute an estimation of the converged solution for time step t^n (step 1). The classical solvers use the initial estimation (step 2) to compute the exact solution following an iterative procedure (step 3). In this layout, a serial coupling strategy is illustrated with the fluid solver being the first participant. Accordingly, the neural network only estimates the part of the solution that is needed by the fluid solver, i.e., boundary displacement. The solid solver receives the initial values at the common interface from the fluid solver accordingly.

of data in the whole three-dimensional simulation domain), which further reduces the cost for both training and evaluation of the DNN. However, it induces a strong need for re-training during the actual simulation as input and output at the coupling interface are not uniquely related (the time step, e.g., also depends on values at the outer domain boundaries of the solid and the fluid). This will be explained in more detail in Sec. 4.2.1.

In the rest, we explain the different steps of the combined neural network and classical solvers simulation. The neural network receives a simulation history sequence (network's input, step 0) to estimate the solution for the new time step (network's output, step 1). The estimated solution is transferred to the classical solvers (step 2) to be used as the first guess to compute the final converged solution (step 3). This procedure is executed until the simulation ends. Note that, in the current study, we follow a serial coupling scheme. Therefore, only the first participant (fluid solver in this work) requires the solution estimation from the DNN.

To generalize the DNN's prediction and ensure that the network gives valid results, i.e., good estimates over many time steps, in particular more time steps than we used for training, we can follow two (naive) approaches to effectively train the neural network. In the first approach, the partial solution history of one simulation can be used as a training data set. For instance, if a simulation requires 1,000 time steps, one can run the first 500 time steps without neural network acceleration, gather training data, train the network and continue the next 500 time steps with acceleration. In the second approach, if we want to generate a network that can be used in a variety of similar scenarios, geometric information of the test scenario (e.g., the tube diameter for the case of a flow in a flexible tube) can be added to the network input, the network can be trained for a few different geometries and used for acceleration for cases with new geometries.

However, both approaches are prone to so-called model-data inconsistency and provide only very limited potential for saving overall computational cost. Training the neural network with one set of data and using it for a completely different setting can result in low estimation accuracy. This is due to the violation of the independent and individually distributed (IID) training and evaluation data assumption. If the distribution of the training and the evaluation data are different, statistical machine learning methods' performance deteriorates. This is a general issue for these methods, details of which can be found in [96]. In the application discussed in this chapter, there is a possibility, that the relation between interface data from one time step to the next (input and output of our network) changes over time and,

second, that the physical solution is not similar enough for the pre-trained DNN to be still applicable if the geometry varies. In such cases, if the estimate solution is not accurate enough, the method can even result in a higher number of subsequent classical iterations. Accuracy is crucially important for convergence acceleration, in particular, to outperform the available numerical methods such as quasi-Newton (explained in Sec. 1.1). We propose on-the-fly re-training of the initial neural network to address this issue.

4.2.1 On-the-Fly Training to Avoid Model-Data Inconsistency

To avoid the degradation of the neural network's estimation over subsequent time steps of the simulation due to the so-called model-data inconsistency, we propose a dynamic training scheme that updates the neural network during the simulation. At given intervals, the last few time steps that have already been computed are used to run a re-training step. This brings the training and the inference data distribution as close as possible and prevents the estimation accuracy drop by re-imposing the IID assumption. A possible downside of this strategy can be the delay due to the re-training step, thus increasing the total simulation time. This delay can be avoided by overlapping the last time step computation and re-training the network.

4.2.2 Neural Network Architecture Design and Hyper Parameters' Tuning

We intend to use neural networks as a surrogate solver to estimate the solution. To obtain an accurate estimation while keeping the network as small as possible, we must consider the physical constraints of the problem when designing the network and tuning the hyperparameters. In a time-dependent continuum domain, as the test case of the current chapter, all unknowns of the system have both spatial and temporal connectivity correlations. Any kind of modeling must account for these connectivity correlations to get accurate results. Note that applying convolution operations when the input data are spatially unstructured is not trivial, since this operation expects the arrangement of the data to be structured to apply the convolution kernel. For unstructured data, one can use graph-based neural networks to apply similar convolutions, see, e.g., [97].

We use a combination of convolutional (CNN) layers and recurrent (RNN) layers to

preserve the mentioned properties. The network's design is schematically shown in Fig. 4.2. It receives a sequence of field variables (e.g., for three subsequent time steps), more specifically, in a fluid-structure interaction test case, the pressure and the velocity from the fluid solver and the structural deformation from the solid solver as an input to predict the same field variables for the next time step. Both the input and the output of the DNN are limited to the common interface to reduce the training and the inference time and computational cost. Each element of the sequence (each time step) initially enters a block of CNN layers, divided into two subblocks of encoder and decoder, to investigate the spatial connectivity. The output of the CNN block is a sequence with the same dimensions. This is then passed on to a block of RNN layers to analyze the time dependency. The network's output is the predicted field values for the next time step.

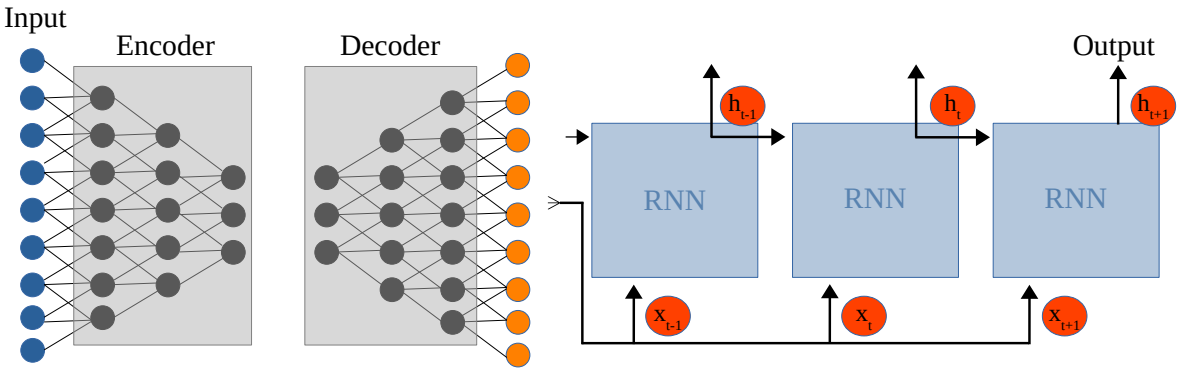


Figure 4.2: Convergence acceleration in partitioned FSI simulation using surrogate neural networks: The architecture of the surrogate neural network for the solution estimation. The input data include a sequence of field variables, i.e., pressure, velocity, and deformation for an FSI test case. These data initially enter two blocks of CNN layers, i.e., encoder and decoder blocks. The output of the CNN blocks is then passed on to the RNN layers. In this figure, RNN layers include three internal sub-layers. Each sub-layer receives one element of the input sequence (x_i) along with the hidden state (h_i) of the previous layer and computes its hidden state. The hidden state of the last sub-layer is the output of the RNN block and the entire network which is the solution estimation for the next time step.

Many parameters must be fitted for efficient training of the neural network. These parameters include but are not limited to, type and number of layers, activation functions, loss function, optimization method, and regularization techniques. For each of these parameters, various options are available in the literature. Our experiments show that the following choices are more efficient than others. We use leaky ReLU activation functions (see Sec. 3.2.3 for details) to map the input of a neuron to its

output. The mean square error is considered as a loss function which is given by:

$$L := \frac{1}{m}\sum_{k=1}^{m}(y_{prediction} - y_{target})^2, \tag{4.1}$$

where y_{target} are the target values and $y_{prediction}$ are the corresponding predictions. To minimize the value of the loss function and compute the network's weights, an ADAM optimizer is incorporated. The numerical values of each field variable (pressure, velocity, and deformation) can have different orders of magnitude. To avoid numerical errors due to this discrepancy, we normalize each variable independently, i.e., all input elements are between -1 and 1 after normalization.

4.3 Parallel Training Based on Training Data Decomposition

Modern scientific and engineering FSI applications are often very complex and require a huge computational effort. Therefore, any simulation software must be able to efficiently run on parallel computers. Efficient parallel training and inference of neural networks are necessary to combine them with classical solvers for a data-integrated simulation. There are various approaches in the literature to parallelize the training and inference of neural networks. These approaches can be generally divided into data parallelization and model parallelization schemes [94]. The first approach divides the data among different processes while the second approach shares all data among processes but distributes the computation among processes. Both approaches require data communication for synchronization. In this section, a novel parallelization scheme for the training of deep neural networks (DNNs) is proposed. The proposed method is compatible with the parallelization techniques that are used for classical solvers.

The proposed method works based on the decomposition of the individual training data sets into smaller spatial sections as shown in Fig. 4.3 instead of distributing complete data sets among processes. Each section is then assigned to an independent neural network. Thus, each network learns the data for its subdomain and the training phase is communication-free. One way to ensure continuity across the subdomains' interfaces is employing a domain overlapping strategy, i.e., subdomains share data points at their interface with the neighbors.

Training: Suppose that we have a set of 1000 training data sets, each representing a

two-dimensional square domain with 100 data points. The following steps must be taken for training [55]:

1. Split each data set into smaller sections, for example, 4 smaller subdomains, each with only 25 data points.
2. Add a layer of interface data from the neighboring sections such that each subsection gets, e.g., 36 data points.
3. Feed each section into an individual network (see Fig. 4.3).
4. Parallelize the training by assigning one MPI rank to each network.
5. Use an individual cost function and optimization process for each network.
6. Train each network for the specific part of the domain and use it to predict only its section.

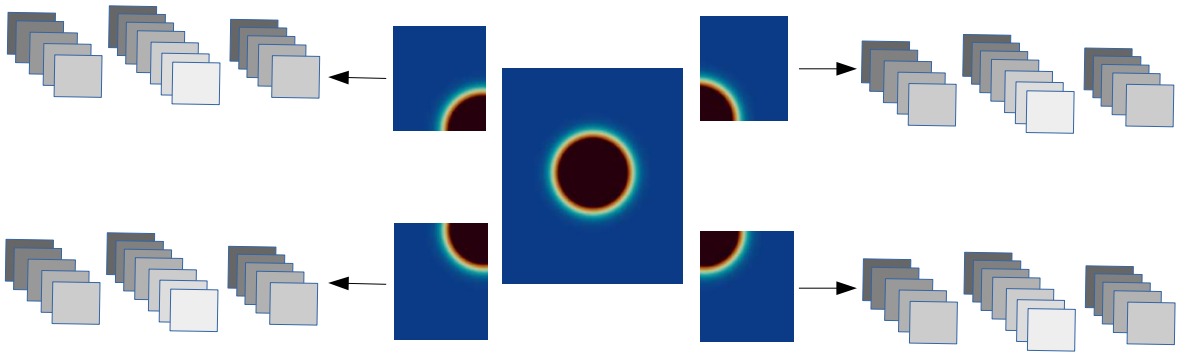


Figure 4.3: Parallelization of neural network training: Each data set is decomposed into smaller spatial sections, each of which is fed into an individual neural network. Individual networks learn only a subsection of the domain, which results in faster training due to fewer data points. In addition, smaller networks with fewer weights can be used, since learning a subsection of the domain is an easier task compared to learning the entire domain. This illustration is taken from [55].

Parallelization of networks that include CNN layers poses further challenges, but also offers options to consider continuity of the output across subdomains. Applying CNNs reduces the size of the two-dimensional input data set by (k-1) rows and (k-1) columns if we use a $k \times k$ convolution kernel. Thus, the network output can not be directly compared to the target data (since the input and the target data have different sizes). For the first layer, the input size can be increased to match its output size with the target data. For our domain partitioning approach, this means, that the input data for neighboring processes must be overlapping. This helps both to match

dimensions and preserve the spatial connectivity between neighboring processes, since the effect of neighboring data points from other subsections can be reflected on the interface data points. In the case of using a single CNN layer, the overlapping input can remove the mismatch of output and target data dimension. For the cases with more than one layer, the following options are possible to solve the dimension mismatch issue:

1. Padding the input data with zeros, to achieve the desired size of the output,
2. padding the input data with data from neighboring subdomains,
3. comparing only the inner $(N - k + 1) \times (N - k + 1)$ data points of the target data to the network output,
4. adding de-convolutional layers or the transpose convolution.

Comparing only the inner data points (option 3) limits the usability of the output data, as substitutes of the actual simulation (data at subdomain interfaces are missing). Therefore, this option is avoided in the current study.

Since each network is responsible only for a single subdomain, there is no need for data exchange between processes in the training phase. The data are directly fed into the network from the memory. This avoids possible bottlenecks due to the data communication.

Inference: Individual networks can be used in parallel for the inference of the subdomain, that they are trained for. Each network receives the input sequence up to time step t to predict the solution at time $t+1$. If the parallel network is integrated into the classical solvers for acceleration, as explained in Sec.4.2, data communication might be needed between processes of the solver and processes of the neural network. This data exchange can be performed directly between the processes that owns the same subdomain to minimize the communication overhead.

4.4 Performance Analysis

To prove the effectiveness of the proposed data-integrated method on improving the performance of FSI simulations, we present a numerical convergence analysis for a fluid-structure interaction test case. We show that the proposed method is able to considerably reduce the number of required iterations. In addition, we prove that the on-the-fly training is able to remove the so-called model data inconsistency

problem. This test case is only a preliminary one to prove the concept and show that the method is able to improve the performance of a simple one-dimensional FSI example. To exploit the method for complex problems, further steps must be performed that will be explained in this section. Furthermore, we investigate the computational performance of the parallel training by means of a single-physics test case.

4.4.1 Data-Integrated FSI Simulation

This section provides numerical evidence for the effectiveness of the proposed data-integrated method using a one-dimensional FSI simulation. This test case studies the fluid flow inside a deformable tube. We initially describe the test case and explain the governing equations for fluid flow and structure deformation along with the coupling equations. Next, we analyze the effect of the proposed method on the convergence speed and the accuracy of the results.

Test Case Description

We present the governing equations for the simulation of an internal flow inside an elastic tube. The test case is schematically shown in Fig.4.4. We briefly explain the one-dimensional model equations for the fluid flow, structure deformation and FSI coupling at the common boundary.

Fluid flow
In the current study, we consider an unsteady and incompressible flow model. Due to the axisymmetric geometry, the flow can be described by quasi-two-dimensional continuity and the momentum equations which read

$$\frac{\partial \mathbf{a}}{\partial t} + \frac{\partial(\mathbf{av})}{\partial x} = 0, \tag{4.2}$$

$$\frac{\partial(\mathbf{av})}{\partial t} + \frac{\partial(\mathbf{av^2})}{\partial t} + \frac{1}{\rho}(\frac{\partial(\mathbf{ap})}{\partial x} - p\frac{\partial \mathbf{a}}{\partial x}) = 0, \tag{4.3}$$

where **a** is the cross-sectional area (dependent on x), **v** is the flow velocity in x-direction, **p** is the fluid pressure, and ρ is the fluid density [98].

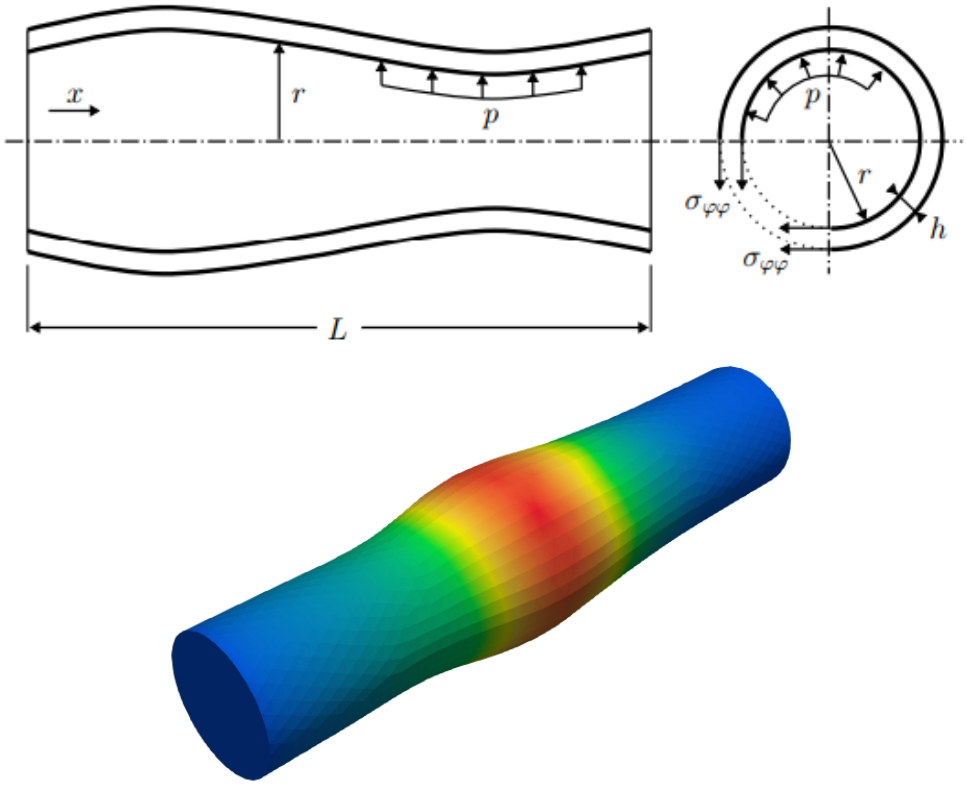


Figure 4.4: One-dimensional tube test case: schematic geometry and parameters. The fluid pressure p acting on the inner tube walls is causing scalar circumferential stress $\sigma_{\phi\phi}$ in the tube walls which leads to deformation in the radial direction. The illustration is taken from [98].

Structure deformation

The conservation laws of mass and momentum govern the structural domain, whose general Lagrangian form can be written as

$$\frac{\partial}{\partial t}\left(\rho_s \frac{\partial \mathbf{d}}{\partial t}\right) = \nabla \cdot \boldsymbol{\sigma}_s, \tag{4.4}$$

where ρ_s is the structural density and $\mathbf{d}$ is the displacement at the reference configuration. The stress tensor $\boldsymbol{\sigma}_s$ is related to the displacement filed. In the current one-dimensional model, it is given by a linear elastic constitutive relation law with the scalar circumferential stress

$$\sigma_{\phi\phi} = E\frac{(r - r_0)}{r_0} + \sigma_0, \tag{4.5}$$

where E is the Young's modulus, r is the tube radius, and $\sigma_{\phi\phi}$ is the circumferential stress with σ_0 at reference position r_0. The motion of the tube wall is, thus, limited

to the radial direction.

FSI coupling condition
The coupling conditions on the fluid-structure interface are derived from the physical equilibrium at the shared boundary. The dynamic FSI interface conditions are given by

$$pr = \sigma_{\phi\phi} h, \tag{4.6}$$

where h is the thickness of the tube wall.

Both the fluid and solid domain are discretized into 100 one-dimensional linear elements with equal size. This implies (i) we have matching meshes at the common interface, (ii) the common interface coincides with the volume of the domains. The time discretization for the fluid and the structure equations is both done by a first-order backward Euler scheme with time step size $\Delta t = 0.01s$. A transient dimensionless inflow velocity is prescribed at the inlet as

$$U_{in} = U_0 + \frac{U_0}{100} sin^2(\pi \frac{n}{T}), \tag{4.7}$$

where n is the time step number, $T = 10$ is the oscillation period of the inflow velocity and $U_0 = 10m/s$ is the initial velocity. The pressure at the outlet is 0. In addition, initial values for velocity, pressure, and deformation are 0. Complete details about this test case can be found in [98].

Numerical Convergence Acceleration Investigation

To accelerate the equation coupling between the structure and the fluid domain, we train a single neural network as a surrogate solver to estimate the solution at the common interface for both domains. The network consists of CNN and RNN layers as explained in Sec 4.2.2. We stack one vector for each domain variable, e.g., pressure, velocity, and deflection, to build the two-dimensional input data. This is shown in Fig. 4.5. Each row corresponds to one domain variable and each column represents data that belong to one point in the mesh. The CNN block (Fig. 4.2) consists of convolutional (encoder) and deconvolutional (decoders) layers. We use a kernel with size 3×3 to mimic the spatial connectivity. With this kernel size, we can capture the effect of domain variables of all neighbors on the target point. This also ensures including the effect of different domain variables on each other. In simple words, with the

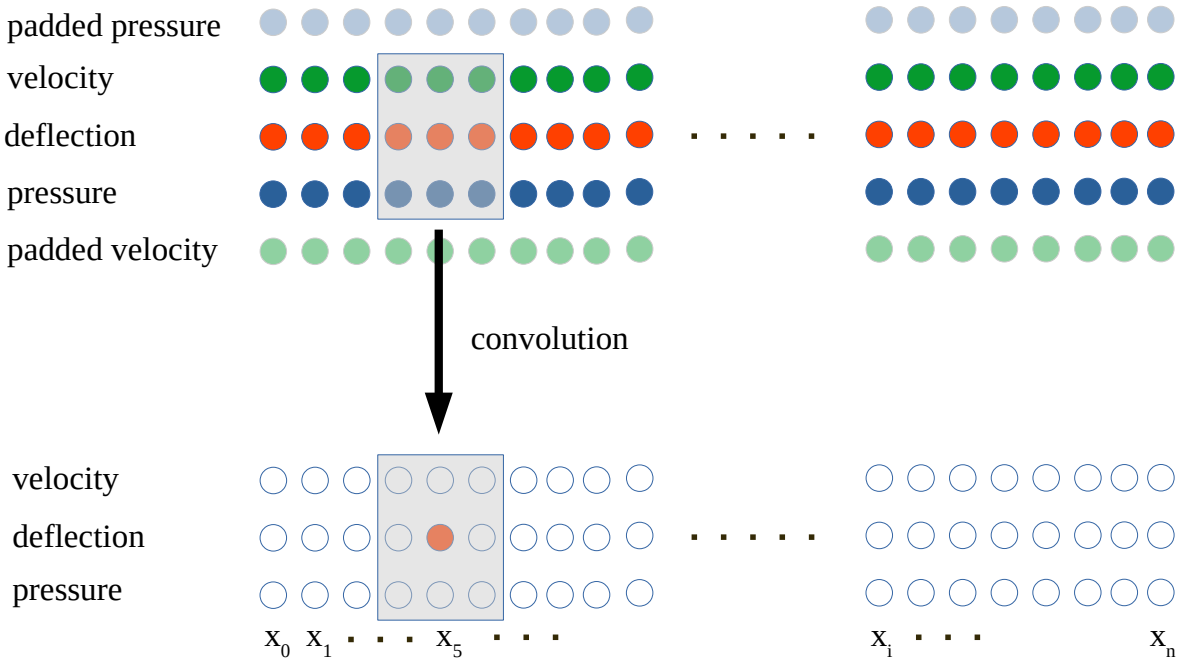


Figure 4.5: One-dimensional tube test case: The input data consists of 3 vectors, each representing one domain variable. In this figure, velocity data are shown in green, deflection in orange, and pressure in blue. The horizontal position of each point (x_i) shows the corresponding grid point in the discretization. The convolution is applied to ensure not only the spatial connectivity between neighbors but also to capture the connection between the domain variables. To ensure that the convolution is applied properly on velocity and pressure data points, we pad the upper and the lower boundary of the input vectors using the pressure and velocity variables, respectively. The padded data are shown with a transparent color.

current data configuration, the convolution considers the effect of the pressure, the velocity, and the deflection of all neighbors on the variables of the target point (for instance, we observe the effect of neighbor points on the predicted deflection of x_5 in Fig. 4.5). Using deconvolution layers, that reverse the effect of convolution, preserves the size of the input data, and makes the padding unnecessary. The output of each layer is mapped using a ReLU activation function before entering the next layer. We use a single RNN layer afterward with 3 hidden layers to learn the temporal connectivity. The network design information is summarized in Table 4.1.

The network input consists of sequences of 3 consecutive time steps (the input consists of time steps $n-2$, $n-1$, n for output time step $n+1$). Each time step of the input initially goes through the CNN layers independently to learn the spatial connectivity. The processed sequence enters the RNN layer for temporal connectiv-

Table 4.1: Deformable tube test case: Neural network design and hyper parameters.

Layer Type	No. of layers	Input seq. size	Kernel size	Act. function
Convolution (encoder)	2	-	3×3	ReLU
DeConvolution (decoder)	2	-	3×3	ReLU
RNN	3 (hidden)	3	-	-

ity learning. The RNN's output is compared with the next time step (the one after the input sequence) via the predefined MSE loss function to compute the network's weight using the ADAM optimization method with a learning rate of 10^{-3}.

The training is performed in two steps: (i) initial training with the first half of the time steps generated with classical simulation to obtain the pre-trained network, (ii) re-training during the hybrid classical–NN simulation after every five time steps to improve the prediction's accuracy. For the initial training, the first 80 time steps are used as the training data set. These time steps can form 77 training sets ($n_{total} - n_{seq}$ with $n_{total} = 80$ being the total number of time steps and $n_{seq} = 3$ the input sequence size). The re-training is performed using only the last 5 time steps. The re-training is faster since fewer input data must be analyzed and fewer training epochs are needed as the network is already pre-trained. The training error given the performed number of training epochs is depicted in Fig. 4.6 for both the initial network training and on-the-fly re-training. First, we observe fluctuations in the error for the initial network training. It seems that, this is due to the learning rate. When the loss decreases, performing a stochastic gradient descent step can increase the error if the learning rate is not small enough. This can be alleviated by reducing the learning rate. However, a smaller rate increases the number of required epochs and the training time. Our experiments show, that reducing the learning rate flattens the error curves, but it does not affect the final training error. Therefore, we did not reduce the learning rate and only truncated the training when the error reached the value of 10^{-6}, which seems to be the smallest possible value in this setup. Further investigations are necessary to better understand and address these error peaks.

In addition, it can be seen, that the network's training error reduces very quickly in the re-training. As expected, the pre-trained network learns a small time window of the simulation much better than a large window. Our experiments show, that even with 1,000 training epochs, the initial network is not able to produce an estimation that is close enough to the actual solution to reduce the number of classical iterations.

On the other hand, the on-the-fly re-training is able to compute a satisfactory estimation after approximately 50 training epochs using only the last five time steps. This estimation is close enough to the actual solution and reduces the number of classical iterations.

In the rest of this section, we present the numerical investigation results on the convergence acceleration of the FSI test case. We initially compare the accuracy of the simulation with and without neural network acceleration in Fig. 4.7 to prove that combining the neural network solver with numerical solvers does not affect the simulation results. The comparison is performed only for the part of the simulation that was not used for training the initial network. The comparison shows, that the acceleration keeps the simulation results unchanged. This is expected, since the classical iterative solution procedure between numerical solvers, after the estimation is provided by neural network, guaranties the correctness of the solution.

We run the same simulation with various acceleration schemes, including Aitken under relaxation, quasi-Newton, and neural network with and without on-the-fly training to compare the number of required iterations for a few time steps. The convergence is checked for the pressure and the deflection as explained in Sec. 1.1. The convergence criterion is set to 10^{-5} for all schemes. For quasi-Newton, the initial relaxation is set to 0.01 and the maximum number of used iterations to 50. In addition, QR2 filter type [32] is used as explained in Sec. 1.1. For the Aitken method, an initial relaxation of 0.1 is used. In addition, since the same discretization is used for both domains, the meshes are matching and we use the nearest neighbor method for the data mapping. The results are provided visually in Fig. 4.8 and numerically in Table 4.2. It is observed, that the acceleration with the neural network has the best performance when the network is re-trained on-the-fly. It is apparent that, without re-training, the prediction's accuracy drops rapidly, and the number of the required subsequent classical iterations increases. On-the-fly re-training updates the network's weight and keeps the prediction accurate enough to minimize the number of required iterations. Note, that the convergence analysis is provided for the first five time steps after the time steps used for training the initial network. However, for the rest of the simulation, we observed a similar pattern.

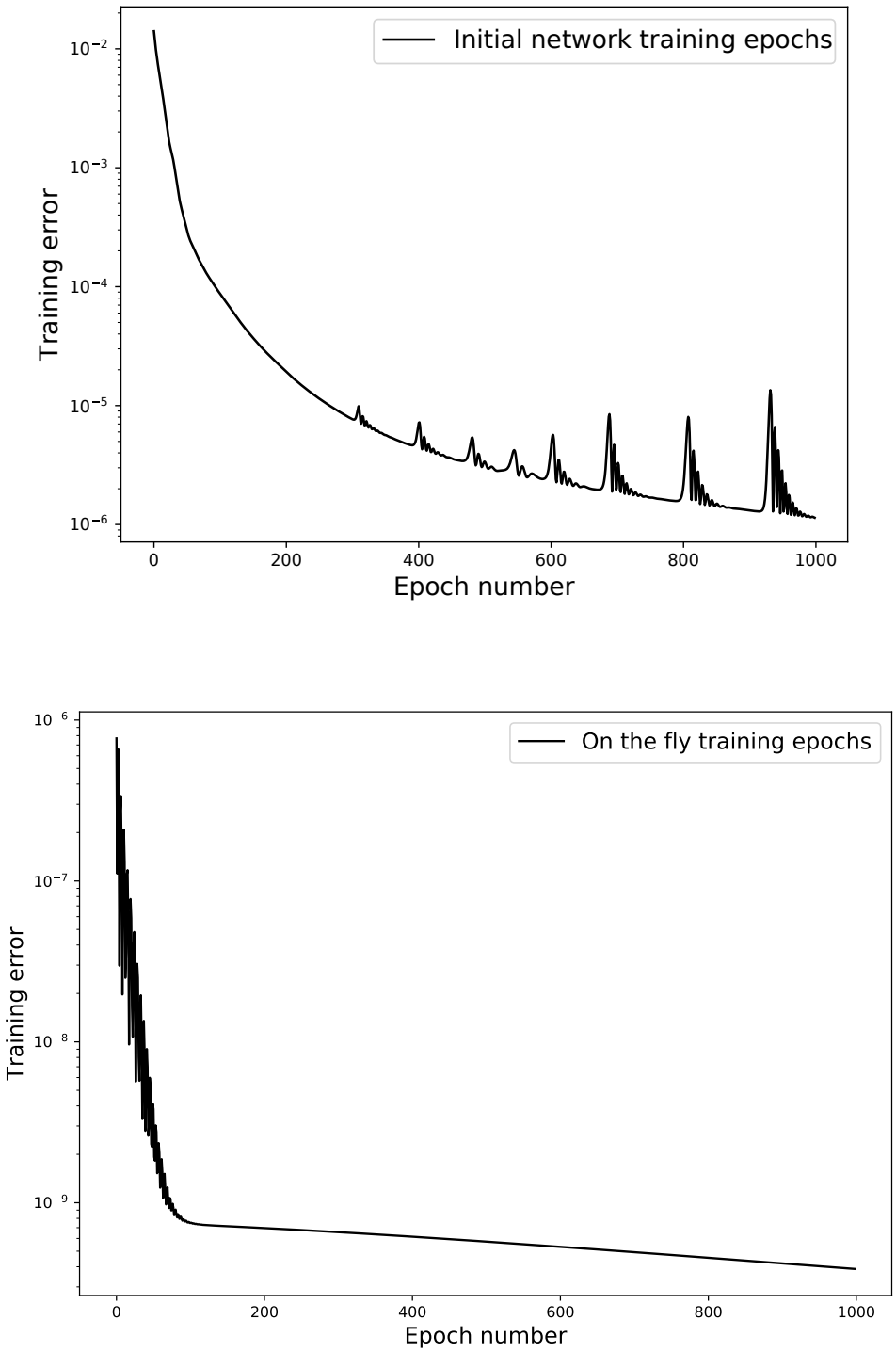


Figure 4.6: One-dimensional tube test case: number of training epochs for initial network training and on-the-fly re-training. The re-training requires significantly fewer epochs to be able to compute a satisfactory solution estimation. The re-training is performed using the first 5 time steps that are computed after the time steps for initial training.

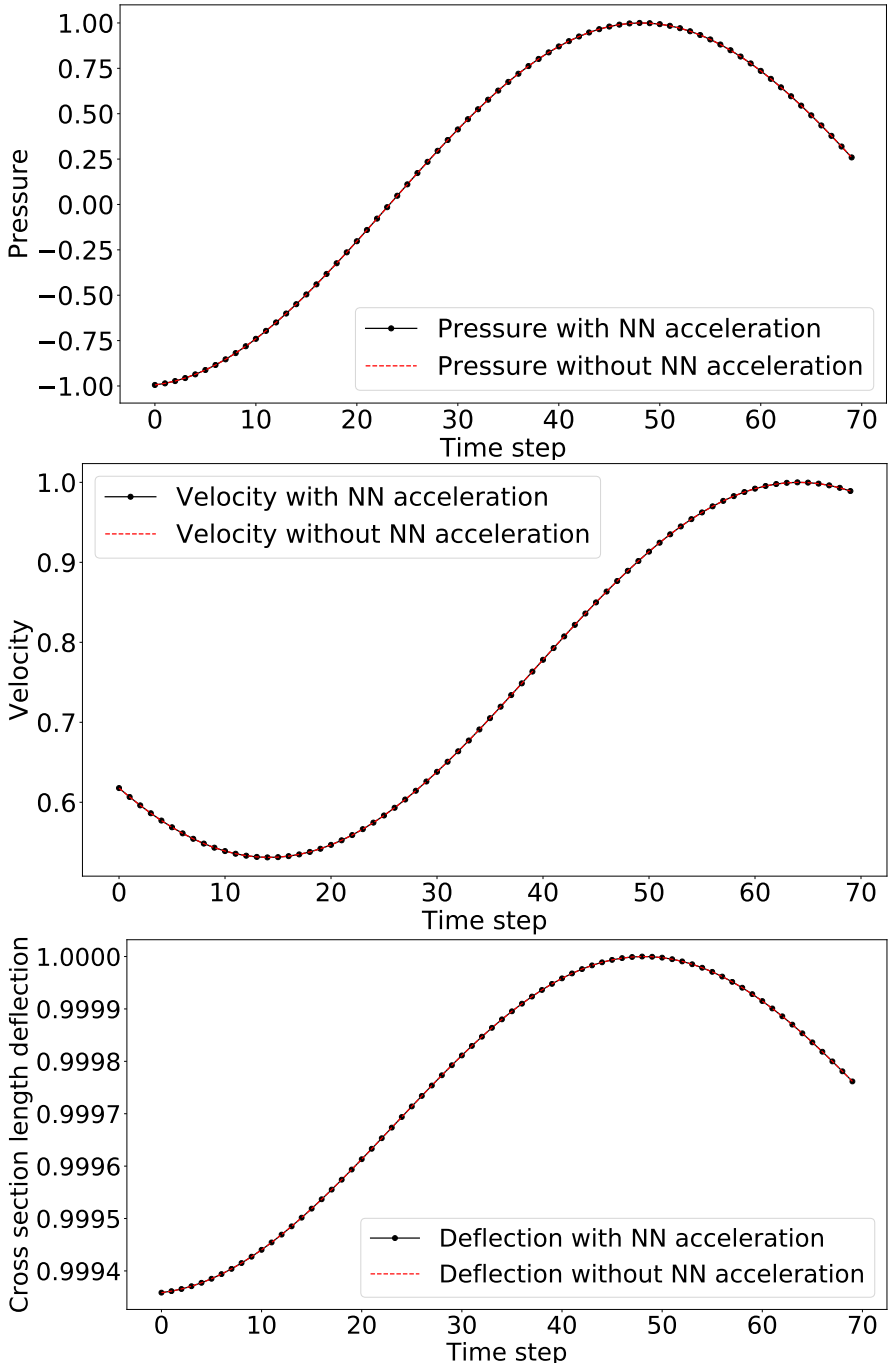


Figure 4.7: One-dimensional tube test case: comparison of simulation results with and without neural network acceleration. The simulation results are compared for cross-section deflection, the fluid pressure, and the fluid velocity at the mid-point of the domain. The results are provided for the part of the simulation (second half), where we applied the neural network acceleration. The first half of the simulation is used to train the initial network. Thus, it is not included in the comparison. The plots are normalized (all values are between -1 and 1 in all three plots) to enable accuracy comparison among variables, i.e., pressure, velocity, and deflection. The black lines represent the simulation with acceleration, while the red ones stand for the case without neural network acceleration. The comparison shows that the acceleration does not affect the simulation results.

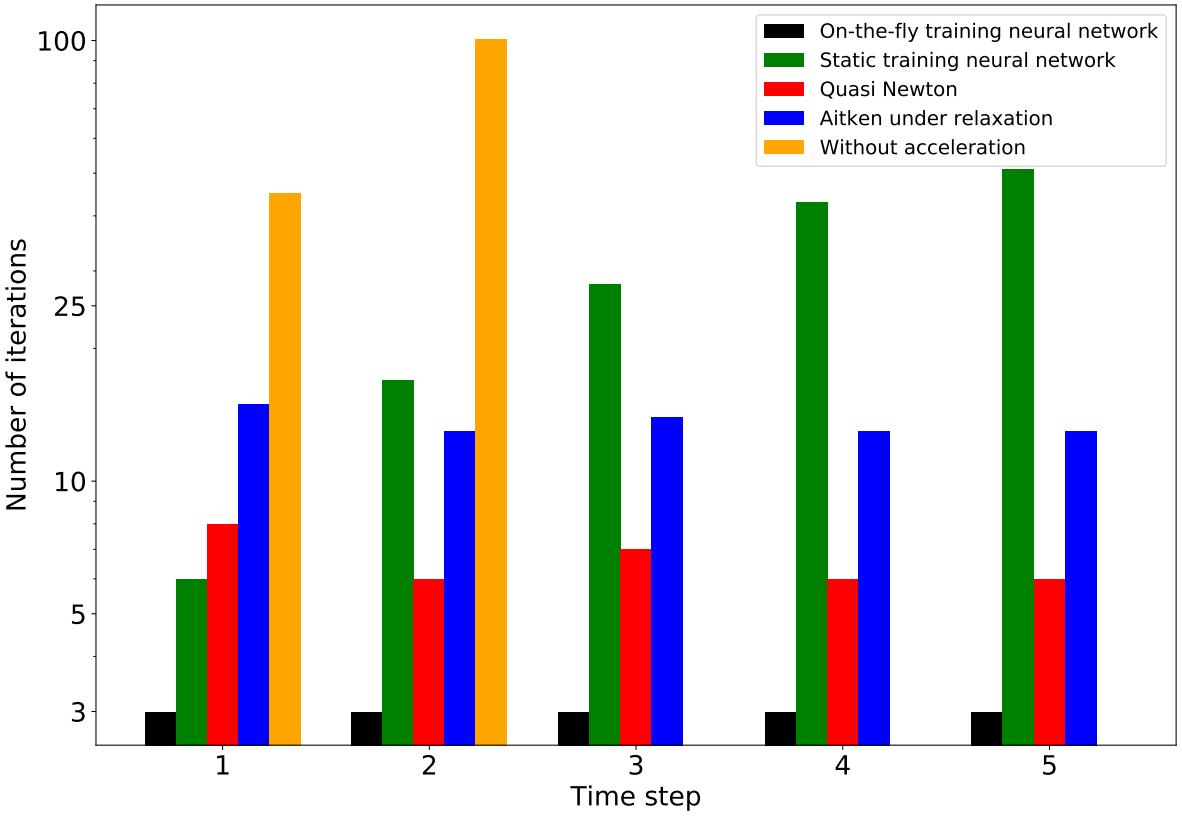


Figure 4.8: One-dimensional tube test case: The required number of iterations is compared for various accelerations schemes, including Aitken under relaxation, quasi-Newton, and neural network with and without on-the-fly re-training. The comparison is performed for the first five time steps after the time steps that are used for initial training. We observe that, without re-training, the neural network is not able to provide an accurate solution estimation after the first time step. The estimation deviates substantially from the actual values, which results in a high number of iterations afterward. The on-the-fly training addresses this issue by updating the neural network's weight every five time steps using the five most recent computed time steps. As seen in Fig. 4.6, approximately 50 epochs are required for the re-training. We observe, that this approach requires the minimum number of iterations among all schemes.

Table 4.2: Deformable tube test case: number of required iterations for various acceleration techniques.

Time step	1	2	3	4	5
No acceleration	45	101	-	-	-
Aitken under-relaxation	15	13	14	13	13
Quasi Newton	8	6	7	6	6
NN acceleration	6	17	28	43	51
NN acceleration with on-th-fly re-training	3	3	3	3	3

4.4.2 Parallelization of Neural Networks Training for Machine Learning of Partial Differential Equations

This section provides numerical performance results for parallel training of neural networks. For performance analysis, we use a two-dimensional test case that investigates the Gaussian pressure pulse within a square domain. The pulse is initially located in the square center and it moves towards boundaries during the simulation. We provide strong scalability measurements and accuracy investigation to prove, that the proposed method can efficiently parallelize neural network training. The parallelization of training is applied to a single-physics problem and is not yet integrated into classical simulation solvers for neural network acceleration of partitioned fluid-structure interaction simulations.

Test Case Description

For our numerical tests, we consider the linearized Euler equations (Eq.(4.8)). The equations to be solved allow determining the perturbation (marked with ′) given a known constant background (denoted with a subscript c) [99]

$$\partial_t \rho' + \nabla \cdot \underbrace{(\boldsymbol{u_c}\rho' + \rho_c \boldsymbol{u'})}_{:=\boldsymbol{m_u}} = 0 \tag{4.8a}$$

$$\partial_t\, \boldsymbol{u'} + \nabla \cdot \left(\boldsymbol{u_c}\boldsymbol{u'} + \frac{1}{\rho_c} p'\right) = 0 \tag{4.8b}$$

$$\partial_t\, p' + \nabla \cdot (\boldsymbol{u_c} p' + \gamma p_c \boldsymbol{u'}) = 0, \tag{4.8c}$$

where ρ, p, and u are the density, pressure, and velocity, respectively. All terms that are non-linear in the perturbations are neglected.

At all four boundaries, outflow boundaries are considered, i.e., the pressure perturbation is set to zero, while all other quantities (density and velocity) have homogenized Neumann boundary conditions.

Initially, the fluid is at rest, the density perturbation is set to zero. The background pressure is $1\ bar$ and the background density is defined to be $1\ kg/m^3$. A pressure perturbation can be prescribed which corresponds to the mentioned Gaussian pulse. For this purpose, we locate a Gaussian pressure pulse in the center of our square domain at P(0.0, 0.0). The half-width of the pulse is set to $0.3\ m$ and the amplitude is 0.5. The background velocity in both directions is zero. For comparison and to train the neural network used in this work, the solver $Ateles$ [84, 65] is used to generate the training and validation data.

Numerical Performance and Accuracy Analysis

In the current test case, each training data set consists of grid-type data points, 256 at each direction accumulating to a total of 65,536 points. For each point, the networks receive density, pressure, and velocity in x- and y-direction. The training data are gathered as a list of three-dimensional arrays, where the x- and y-direction corresponds to the domain and the z-direction accounts for the various data types (density, pressure, and velocities). For parallelization, we decompose the domain into smaller subdomains as described in Sec. 4.3 and use an individual network for each subdomain.

The network that is used for each subdomain of this test case consists of convolutional layers to mimic the spatial connectivity and accounts for the effect of domain variables on each other. The output of each layer is mapped using a ReLU activation function before entering the next layer. The network's output is compared with the next time step via the predefined MSE loss function to compute the network's weight. We use the ADAM optimization method with a learning rate of 10^{-3}. The network design information is summarized in Table 4.3.

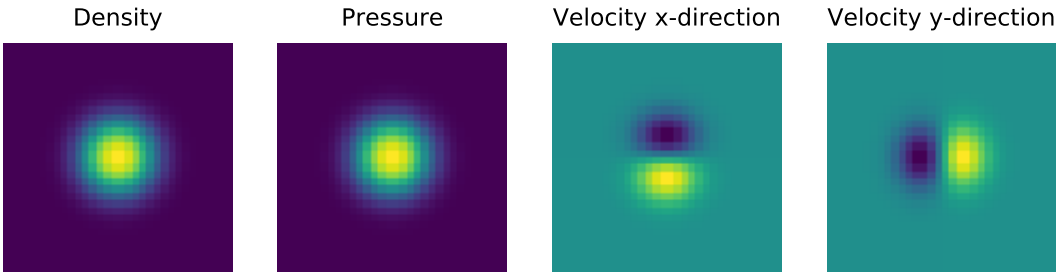
We initially compare the network's output for parallel and serial training in Fig. 4.9 to investigate the effect of parallelization on the accuracy of predictions. For both cases, the network receives the domain information, including pressure, density, and velocities at time step (t), and predicts the domain status in the next time step. In total, 1500 training and validation data are produced by running a single simulation. The first 1000 time steps are used for training and the remaining ones for validation.

Table 4.3: Neural network training parallelization: CNN layers architecture and arrangement. Output channels of each layer must match with the input channels of the next layer.

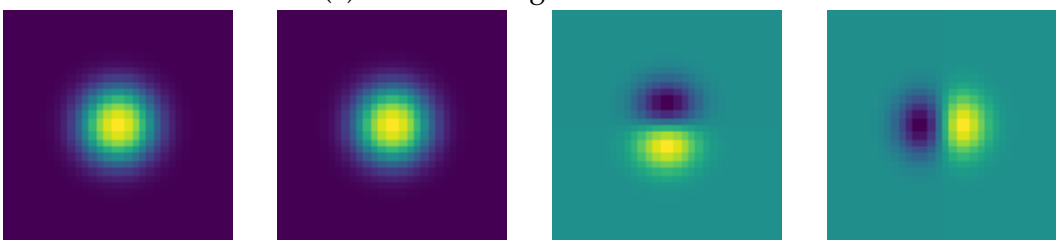
layer number	input channels	output channels	kernel size	padding
1	4	6	$4 \times 6 \times 3 \times 3$	Yes
2	6	16	$6 \times 16 \times 3 \times 3$	Yes
3	16	6	$16 \times 6 \times 3 \times 3$	Yes
4	6	4	$6 \times 4 \times 3 \times 3$	Yes

For the parallel training, the domain is decomposed into four square subdomains and each is assigned to a separate neural network. The domains are overlapping to minimize the possible discontinuity in the common interfaces. It seems that, for the current test case, we can preserve the continuity across the common interface by overlapping the subdomains. For more complicated geometries, further investigations are required to study the effect of parallelization on the accuracy of the network's prediction.

In the rest of this section, we present a strong scalability analysis for the proposed scheme up to 64 CPU cores in Fig. 4.10. The domain is decomposed into square subdomains and each subdomain is assigned to a separate network. Each network is trained independently and the training is stopped when the training accuracy reaches a certain value (in this case to 10^{-5}). We observe an almost perfect strong scaling, where the training time reduces as the number of CPU cores is increased. This behavior is expected, as parallelization reduces the size of training data and thus the training time. In addition, avoiding communication during training contributes to the observed efficiency. It must be noted, that the inference time is very small since it is performed only for a single time step.



(a) Serial training and inference.



(b) Parallel training and inference.

Figure 4.9: Neural network training parallelization: comparison of the neural network's output for the parallel and the serial inference. The input and output data are compared for the time step 25 in the validation data set. For the parallel inference, the domain is decomposed into four smaller square subdomains. A layer of interface data with the width of 2 data points from the neighboring subdomains is added to overlap the subdomains. This overlapping is intended to ensure continuity across the common interfaces of the subdomains. It seems that, for the current test case, the overlapping strategy can effectively preserve the continuity for all domain variables.

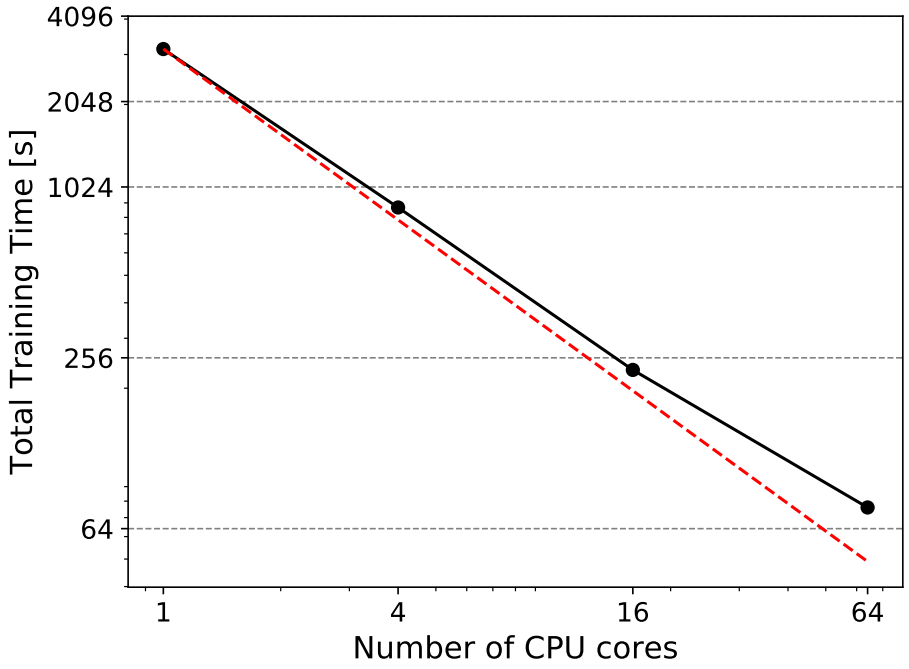


Figure 4.10: Neural network training parallelization: strong scalability measurements for the proposed parallelization scheme. Each training data set is split into smaller square subdomains. Each subdomain is assigned to a separate neural network. The training is performed for each network until the training error of all networks is reached to 10^{-5}. The training time decreases as the number of exploited CPU cores increases.

4.5 Summary

We presented a data-integrated simulation environment for the partitioned solution of strongly coupled FSI problems. The partitioned approach provides the opportunity to reuse available single-physics solvers. This reduces the software development cost, however, for problems in which the subdomains are strongly coupled, the method requires a high number of iterations to reach a converged solution.

In the current study, we combined deep neural networks with classical solvers for a more efficient simulation by reducing the number of required iterations. The deep neural network, consisting of CNN and RNN layers, is trained to provide a solution estimation at the beginning of each time step. The classical solvers benefit from the estimation of the converged solution and need fewer iterations to reach the final solution. We introduced a dynamic re-training scheme that keeps the network updated during the simulation and avoids the so-called model-data inconsistency. The method improves the performance of the simulation while keeping the solution accuracy untouched. For a simple, but representative, one-dimensional test case that studies the fluid flow within a deformable tube, the data-integrated method reduces the number of iterations to 3, while the purely classical methods, such as quasi-Newton, require at least 6 iterations.

In addition, as a first attempt to prepare the method to be used in more complex problems, we introduced a parallelization scheme for the training of neural networks that can be used in such applications. The method partitions the simulation domain, similar to parallelization methods in classical simulations, and assigns each subdomain to an individual neural network. We show that the training can be perfectly scaled to many processors (64 CPU cores in our test case).

Another challenge that must be addressed before applying the method to more complex simulations is the capability of training the network with data provided on unstructured grids. Modern simulation software use unstructured grids to generate computational mesh for complex geometries. Application of classical convolution to these data structures is not possible, since the operation can be applied only on structured grids. A possible remedy for this problem is using graph-based neural networks which are capable of handling unstructured data. This topic will be investigated in future works. In addition, in a three-dimensional test case, the DNN must be able to perform the estimation based on partial data from the domain, i.e., only data from the common interface. This requires further investigations since not having access to a major part of the domain can reduce the estimation accuracy.

5 A Highly Scalable Solver with GPU Acceleration for Partitioned Solution of Fluid-Structure Interaction Problems

This chapter presents a highly scalable solver with GPU acceleration capability for the partitioned solution of the fluid-structure interaction (FSI) problems. This solver represents a showcase demonstrating how hybrid CPU-GPU architectures can be efficiently used for surface-coupled multi-physics simulations with minimal additional implementation cost based on existing solvers and a parallel coupling software. An adaptive off-loading scheme is introduced to port the solvers' most compute-intensive kernels to GPUs. I.e., the single physics solvers are modified to be capable of exploiting hybrid architectures (CPU-GPU) efficiently, while the coupling itself is not affected and is still executed on the CPUs. A multi-step data-driven load balancing approach based on the scheme presented in chapter 3 is derived to efficiently distribute the available computational resources among the solvers.

We show scalability and efficiency results for a patient-specific aorta test case on a CPU-only and on a hybrid CPU-GPU machine. The results show strong scalability for up to 13,440 cores on the CPU-only machine and for up to 8 compute nodes (232 CPU cores and 16 GPUs) on the hybrid system. In addition, the overall run time of the coupled FSI simulations is reduced by a factor of 7.2 using GPU acceleration, compared to the case with the same number of CPUs without GPU acceleration. In the remainder of this chapter, we briefly introduce the FSI problem and available solvers addressing this sort of problem in Sec. 5.1. Section 5.2 explains the governing equations and numerical methods for each sub-problem along with the coupling conditions and convergence acceleration techniques. Section 5.3 explains the single physics solvers' parallelization to exploit hybrid CPU-GPU architectures. Section 5.4 describes the inter-solver parallelization which includes parallel data communication and a multi-step inter-solver load balancing scheme. Scalability and efficiency measurements and analysis are presented in Sec. 5.5, while Sec. 5.6 summarizes and concludes the chapter.

5.1 Introduction

The interaction of a flexible structure with a flowing fluid is observed in various physical phenomena [100] with a wide range of applications in many fields of engineering, such as the stability of aircraft wings subject to turbulent flow [101], the blood flow in the arteries [102], the effect of winds on bridges [103], the vibration of wind turbine blades [104], and the vibrations of heat exchangers [105].

Given the high complexity of FSI problems, their investigation requires a huge amount of computational resources and time. To reduce the simulation time, parallel solvers can be used to efficiently exploit distributed systems, e.g. supercomputers. Recently, many efforts have been made to develop efficient and scalable multi-physics solvers for FSI problems using either monolithic [106, 107, 108] or partitioned [109, 29, 110] approaches. In the remainder of this section, I briefly introduce some of the partitioned FSI solvers that have been presented in the literature to exploit parallel computing architectures.

Cajas et al. [29] presented a parallel solver which combines two instances of an in-house code (for fluid and solid). MPI communication is used for a point-to-point inter-solver data transfer, while each solver uses a master-slave communication scheme. An inter-code load balancing method is proposed based on overloading the available cores to minimize idle time. In this method, some of the cores execute MPI processes for fluid and solid solvers alternatingly. The coupled solver scales well up to 1280 MPI processes on 768 CPU cores [29].

Hewitt et al. [110] developed a framework based on coupling open-source single-physics solvers (OpenFOAM for the fluid and ParaFEM for the structure). In this framework, the inter-code communication is handled via two master ranks (each belongs to one of the solvers). The solver is shown to scale well on 1,536 cores for a coupled FSI problem. The coupled solver uses the same cores for the fluid and the solid solver, executing different MPI processes consecutively (serial coupling) for each solver.

In addition, there are several solvers capable of exploiting hybrid architectures for FSI problems in literature. For instance, Jiang et al. [111] presented a GPU-accelerated solver which follows a multi-code coupling strategy for the solution of FSI problems in the field of biomechanics. A lattice Boltzmann solver is used for the incompressible fluid simulation along with an explicit finite element solver for the solid domain. An Aitken relaxation is employed to improve the convergence of the

fixed point coupling iterations. All computations from both solvers are offloaded to GPUs using the CUDA library. The numerical performance analysis shows that the GPU acceleration (with Nvidia GeForce 108Ti) results in 18.44 times faster simulation compared to CPU-only simulation (on Xeon E5-1620v4 CPU) for an FSI benchmark on a single GPU.

In the current work, we present a scalable and efficient simulation environment for the partitioned solution of FSI problems with GPU acceleration capability. We follow a partitioned approach where we couple two instances of the TermoFluids solver [27] through the preCICE coupling library [34]. The CPU-only single-physics solvers for this work and the semi-implicit coupling strategy are provided by Naseri [112, 113]. Coupling the solvers via the preCICE library is also a joint contribution of Naseri and myself [7]. For the current study, we modified the single-physics solvers to efficiently exploit hybrid CPU-GPU architectures. Our code is configured in a way that we exploit one GPU per CPU core. A data-based load balancing scheme, similar to the approach introduced in Chapter 3, is also incorporated to maximize the parallel efficiency. In addition, the inter-solver communication in our solver is limited to CPUs, and data exchange between GPUs is carried out indirectly via CPUs.

5.2 Governing Equations and Numerical Methods

In this section, the fluid and the structure governing equations along with the coupling conditions on their common interface are presented. Moreover, the numerical methods for discretization and solvers in each single-physics problem as well as the iterative coupling method are described. The fluid and the structure domains are referred to as $\Omega_f(t) \subset \mathbb{R}^3$ for all t in $(0, T)$ and $\Omega_s(t) \subset \mathbb{R}^3$ for all t in $(0, T)$, respectively, where $t \in (0, T)$ denotes time. The fluid-structure interface is the common boundary of the domains, denoted by $\Gamma(t) = \partial\Omega_f(t) \cap \partial\Omega_s(t)$.

5.2.1 Fluid Solver

Governing Equations: The unsteady flow of an incompressible viscous fluid is mathematically described by the Navier-Stokes equations. To solve the fluid flow in a

moving domain, an Arbitrary Lagrangian-Eulerian (ALE) formulation is used together with a conforming mesh technique. In a moving domain, the ALE formulation of the Navier-Stokes equations is given by

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{c} \cdot \nabla \mathbf{u} = \frac{1}{\rho_f} \nabla \cdot \boldsymbol{\sigma}_f, \tag{5.1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{5.2}$$

where $\mathbf{u}$ is the fluid velocity and ρ_f the fluid density. The vector $\mathbf{c}$ represents the ALE convective velocity $\mathbf{c} = \mathbf{u} - \mathbf{w}$, which is the relative fluid velocity to a domain moving with a velocity $\mathbf{w}$. For an incompressible Newtonian fluid, the stress tensor $\boldsymbol{\sigma}_f$ can be calculated as

$$\boldsymbol{\sigma}_f = -p\mathbf{I} + \mu_f(\nabla \mathbf{u} + \nabla \mathbf{u}^T), \tag{5.3}$$

where p is the fluid pressure, $\mathbf{I}$ the unit tensor, and μ_f the dynamic viscosity of the fluid.

Numerical Methods: A fractional-step projection method along with an explicit time advancement is used to solve the velocity-pressure coupling of the momentum equation. The following three steps must be performed for the solution of the fluid governing equations from time step n to $n+1$

$$\mathbf{u}^p = \mathbf{u}^n - \Delta t[\frac{3}{2}(\mathbf{c}^n \cdot \nabla \mathbf{u}^n - \frac{\mu_f}{\rho_f}\Delta \mathbf{u}^n) - \frac{1}{2}(\mathbf{c}^{n-1} \cdot \nabla \mathbf{u}^{n-1} - \frac{\mu_f}{\rho_f}\Delta \mathbf{u}^{n-1})], \tag{5.4}$$

$$\frac{\Delta t}{\rho_\mathrm{f}} \Delta p^{\mathrm{n+1}} = \nabla \cdot \mathbf{u}^{\mathrm{p}}, \tag{5.5}$$

$$\mathbf{u}^{\mathrm{n+1}} = \mathbf{u}^{\mathrm{p}} - \frac{\Delta t}{\rho_\mathrm{f}} \nabla p^{\mathrm{n+1}} \tag{5.6}$$

in Ω_f^{n+1}, where Δt is the time increment and $\mathbf{u}^p$ is a predicted velocity field which does not satisfy the incompressibility condition (Eq. (5.2)). We use an explicit second-order Adams-Bashforth method for the temporal discretization of the convective

and diffusive terms in Eq. (5.4). Using an explicit method avoids solving a nonlinear system of equations for the fluid velocity field.

The fluid pressure field is obtained by solving a Poisson equation as in Eq. (5.5). This pressure field is then used to project the intermediate velocity field onto a divergence-free field through the correction in Eq. (5.6). The Poisson equation for the pressure is the only implicit part of the discretized fluid equations, for which a linear system of equations is constructed and solved using a conjugate gradient solver with a Jacobi preconditioner.

The fluid equations are discretized in space using a finite-volume method on a collocated unstructured mesh with second-order symmetry-preserving schemes. These schemes conserve the mass, momentum, and kinetic energy of the flow at the discrete level which is crucial in turbulent flow simulations [114, 115].

A conforming mesh technique is used to track the moving boundary. Thus, the fluid mesh needs to move to adapt to the new location of the interface. A parallel moving mesh technique based on the radial basis function interpolation method [116] is applied to move the fluid grid in accordance with the new location of the interface and define the discretized fluid domain at the new time step Ω_f^{n+1}. The method evaluates an interpolated position for the interior vertices of the fluid grid given the known displacement on the interface. Since the method does not need mesh connectivity information, it can be applied to both structured and unstructured grids. Moreover, it only requires solving a linear system of equations whose size is limited by the number of vertices on the fluid-structure interface.

The domain velocity field $\mathbf{w}$ is evaluated using the Geometric Conservation Law (GCL) [117]. The GCL guarantees the conservation of volume during the mesh translation. For any control volume (CV) in the fluid domain, the GCL reads

$$\frac{\partial v}{\partial t} - \int_s \mathbf{w} \cdot \mathrm{d}\mathbf{A} = 0, \tag{5.7}$$

where v and s stand for the volume and the boundary surface of the CV, respectively, and $\mathbf{A}$ is the area vector pointing outward. In the discretized domain, the time rate of change of volume of a CV is equal to the sum of volumes swept by its faces. We evaluate the domain velocity at each face of the control volume ($\mathbf{w}_{face}$) based on the volume swept by that face. With a second-order backward discretization, it reads

$$\mathbf{w}_{face}^{n+1} = \frac{3}{2}\left(\frac{\delta v}{A\Delta t}\mathbf{n}\right)^{n+1} - \frac{1}{2}\left(\frac{\delta v}{A\Delta t}\mathbf{n}\right)^{n}, \tag{5.8}$$

where A is the surface area, $\mathbf{n}$ is the unit normal vector of the face, and δv is the volume swept by the translated face at one time step.

5.2.2 Structural Solver

Governing Equations: The conservation laws of mass and momentum govern the structural domain. Their Lagrangian form can be written as

$$\rho_s^0 = \rho_s J, \tag{5.9}$$

$$\frac{\partial}{\partial t}\left(\rho_s \frac{\partial \mathbf{d}}{\partial t}\right) = \nabla \cdot \boldsymbol{\sigma}_s, \tag{5.10}$$

where the reference (undeformed) configuration of the body is denoted with superscript 0, ρ_s is the structural density, and $\mathbf{d}$ is the displacement at the reference configuration. The hyper-elastic Saint Venant-Kirchhoff constitutive model relates the Cauchy stress tensor $\boldsymbol{\sigma}_s$ to the displacement field:

$$\boldsymbol{\sigma}_s = \frac{\mathbf{B}}{2J}[2\mu_s(\mathbf{B}-\mathbf{I}) + \lambda_s \mathrm{tr}(\mathbf{B}-\mathbf{I})], \tag{5.11}$$

where $\mathbf{B}$ represents the left Cauchy-Green deformation tensor $\mathbf{B} = \mathbf{F}\cdot\mathbf{F}^T$, and μ_s and λ_s are the Lamé's parameters. The material deformation tensor $\mathbf{F}$ is evaluated as $\mathbf{F} = \mathbf{I} + \nabla\mathbf{d}$ and its determinant is denoted by $J = \det(\mathbf{F})$.

Numerical Methods: An implicit trapezoidal rule time integration is used to solve the solid equations:

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \frac{\Delta t}{2\rho_s}\left(\nabla\cdot\sigma_s^{n+1} + \nabla\cdot\sigma_s^n\right), \tag{5.12}$$

$$\mathbf{d}^{n+1} = \mathbf{d}^n + \frac{\Delta t}{2}\left(\mathbf{v}^{n+1} + \mathbf{v}^n\right), \tag{5.13}$$

where $\mathbf{v}$ approximates the first time derivative of the displacements $\mathbf{d}$, $\mathbf{v} = \frac{\partial \mathbf{d}}{\partial \mathbf{t}}$. The equations are discretized in space using a finite-volume method with a total Lagrangian approach. The momentum equation is integrated on the undeformed configuration.

The coupling between different directions of the displacement and the geometrical and material non-linearities found in $\nabla\cdot\sigma_s^{n+1}$ in the right-hand side of Eq. (5.13) are split up into a linear diffusion part $\mathcal{K}\Delta d^{n+1}$ and a non-linear deferred correction.

Thus, to solve a time step, we perform an outer fixed-point iteration (accelerated by an Aitken Δ^2 acceleration technique for multidimensional problems [118])

$$\begin{aligned}\mathbf{d}^{n+1,k+1} &= (1-\alpha_k)\mathbf{d}^{n+1,k} + \frac{\alpha_k\,\Delta t^2}{4\rho_s}\mathcal{K}\Delta\mathbf{d}^{n+1,k+1} + \\ &\quad \alpha_k\left(\mathbf{d}^n + \Delta t\,\mathbf{v}^n + \frac{\Delta t^2}{4\rho_s}\left(\left(\nabla\cdot\sigma_s^{n+1,k} - \mathcal{K}\Delta\mathbf{d}^{n+1,k}\right)\right)\right),\end{aligned}$$

where α_k is the relaxation factor of the Aitken acceleration.

The inner diffusive system is discretized by a central difference scheme with a non-orthogonal correction. Selecting an optimal value of the diffusion coefficient $\mathcal{K} = (2\mu_s + \lambda_s)$ improves the convergence of the iterative process. The gradient of the displacement (and hence, the strain and stress tensors) are evaluated directly on the cell faces.

5.2.3 FSI coupling

Governing Equations: The coupling conditions on the fluid-structure interface are derived from the physical equilibrium (kinematic and dynamic equilibrium) at the shared boundary. For a no-slip type interface, they read

$$\mathbf{u}_\Gamma = \frac{\partial \mathbf{d}_\Gamma}{\partial t}, \tag{5.14}$$

$$\boldsymbol{\sigma}_s\mathbf{n}_\Gamma = \boldsymbol{\sigma}_f\mathbf{n}_\Gamma \tag{5.15}$$

at Γ, where $\mathbf{n}_\Gamma$ is the unit normal vector on the interface.

Numerical Methods: A Dirichlet-Neumann domain decomposition approach for the coupling of the fluid and the structure equations is followed in this work. Therefore, the fluid equations are solved for a known displacement of the interface (Dirichlet boundary condition derived from kinematic equilibrium Eq. (5.14)), while the structure equations are solved for known stress on the common interface (Neumann boundary condition derived from dynamic equilibrium Eq. (5.15)).

In this work, a semi-implicit FSI coupling method, as proposed in [119, 113], is used. The method segregates the fluid pressure term and strongly couples it to the structure via coupling iterations. The remaining fluid terms and the geometrical

nonlinearities are only loosely coupled to the structure and, therefore, solved only once at every time step. Algorithm 5 describes the method used in this work.

The Segregation of the fluid pressure term is achieved based on a projection method

Algorithm 5 One time step in the semi-implicit FSI coupling method

1: Predict the location of the interface by extrapolating from previous time steps.
2: Move the mesh and evaluate the surface velocities.
3: Solve the fluid ALE convection-diffusion equation for the predicted velocity field (Eq. (5.4)).
4: **while** not converged **do** $\triangleright \begin{pmatrix} \mathcal{S}(\boldsymbol{\sigma}_\Gamma) \\ \mathcal{P}(\mathbf{d}_\Gamma) \end{pmatrix} = \begin{pmatrix} \mathbf{d}_\Gamma \\ \boldsymbol{\sigma}_\Gamma \end{pmatrix}$
5: Solve equations of the pressure (Eq. 5.5) and the structure (Eq. 5.12-5.13).
6: Update interface stresses and velocities.
7: Execute quasi-Newton acceleration on stresses and velocities.
8: **end while**
9: Evaluate the corrected velocity field using the converged pressure field (Eq. (5.6)).
10: Apply the boundary condition on the corrected velocity using the converged deformation.

to solve the fluid equations. From the discretized fluid equations at Eq. (5.4) to Eq. (5.6), only the Poisson equation for pressure (Eq. (5.5)) is strongly coupled to the structure. Hence, this equation is solved several times per time step, i.e., in each coupling iteration, while the remaining fluid equations and the mesh movement are only executed once per time step. In addition, the Poisson equation for the pressure is the only implicit part of the fluid discretized equations, for which a linear system of equations is solved. Therefore, the solution of this equation represents the majority of the computational cost of solving the fluid equations in the coupled FSI problem. The structure equations are implicitly discretized in time as described in Eq. (5.14). Each time step of the coupled problem is solved by performing mesh movement, convection, and diffusion in the fluid solver and, subsequently, solving the fixed point equation

$$\begin{pmatrix} \mathcal{S}(\boldsymbol{\sigma}_\Gamma) \\ \mathcal{P}(\mathbf{d}_\Gamma) \end{pmatrix} = \begin{pmatrix} \mathbf{d}_\Gamma \\ \boldsymbol{\sigma}_\Gamma \end{pmatrix} \tag{5.16}$$

iteratively, where $\mathcal{S}$ represents the mapping of interface stress tensor $\boldsymbol{\sigma}_\Gamma$ to interface displacement vector $\mathbf{d}_\Gamma$ by the structure solver and $\mathcal{P}$ the mapping of interface

displacement to interface stress by solving the pressure Poisson equation and recalculating stress in the fluid solver. The convergence of the iterative solution of Eq. (5.16) is accelerated by using a multi-vector quasi-Newton method, as explained in Sec. 1.1.

The solution of the pressure Poisson equation for the fluid (Eq. 5.5) is where we use the GPUs for acceleration, as described in the following sections.

5.3 Single-Physics Solver Parallelization and GPU Acceleration

In a partitioned simulation, two levels of parallelization can be exploited: (i) the intra-solver level and (ii) the inter-solver level. While the former corresponds to the efficient implementation of single physics solvers, the latter concerns the optimal inter-solver load balancing, the implementation of efficient high-speed communication between them, and efficient data mapping between non-matching meshes. This section focuses on (i), i.e., the parallelization of solvers and the GPU acceleration of the fluid solver.

5.3.1 Single-Physics Solver Parallelization

This section summarizes the parallelization for the single-physics solver which is explained in more detail in [7]. A distributed-memory model is used for parallelization of the computations at each single-physics solver and communication between processes is established using the Message Passing Interface (MPI) standard. The distributed-memory parallelization is carried out based on spatial domain decomposition. We decompose the computational domain Ω (either Ω_f or Ω_s) into n non-overlapping subdomain blocks, $\Omega_0, ..., \Omega_{n-1}$, and assign each block to a different MPI rank. Each block contains cells (control volumes) that are owned by the MPI rank, and halo cells that represent the neighbors of the owned cells belonging to a different process. Two cells are considered to be neighbors if they share one or more vertices. Figure 5.1 schematically shows a discretized domain and its decomposition into two subdomain blocks.

The METIS library [120] is used to carry out the decomposition of the computational domain. METIS divides the computational mesh into roughly equal partitions using a parallel multilevel $k\text{-}way$ graph-partitioning method to minimize both the load imbalance and the number of halo cells [120]. Roughly equal sizes of the blocks

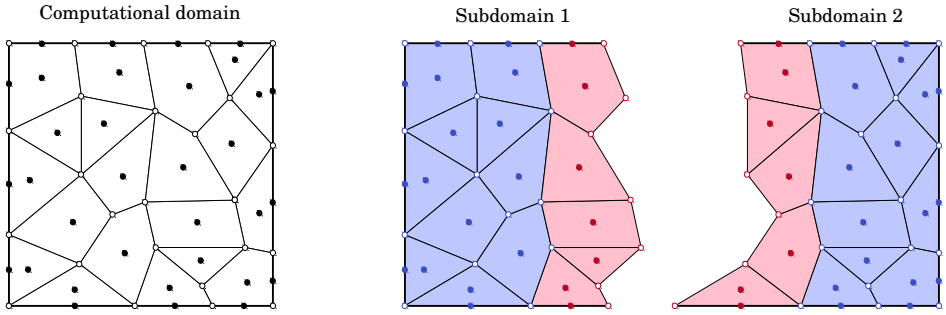


Figure 5.1: Spatial domain decomposition for a single-physics solver: a discretized domain (left) is decomposed into two subdomain blocks (right). Cell and boundary nodes are represented as filled circles and vertices as empty circles. The owned elements (cells, nodes and vertices) of each process are shown in blue while the halo elements are shown in red. This illustration is taken from [7].

balance the computational load among the processors while minimizing the number of halo cells reduces the necessary data communication.

The parallel efficiency is mainly limited by inter-process communication. This communication in the implementation of this work can be generally divided into two categories: (i) global reduction operations which we use to evaluate norms, dot products, and to obtain global extrema (e.g., in time step evaluation); (ii) parallel point-to-point communication between the processes in order to update data in the halo cells. (i) is rarely required in the solver and is carried out by simply calling the corresponding `MPI` collective operations. For (ii), i.e., to update halo data, the non–blocking functions `MPI_Isend` and `MPI_Irecv` are used followed by a `MPI_Waitall` function for synchronization called by all processes. The communication initialization is carried out only once at the beginning, in which a mesh analysis is performed to find and store the list of connected processors and the list of data that must be communicated for each MPI rank. During run time, a loop is created over this small list to invoke the respective communication.

5.3.2 GPU Acceleration of the Fluid Solver

The respective discretized systems of equations of both the fluid flow and the structure solver are sparse. Therefore, the computations have low arithmetic intensity and are memory-bound. Already for CPUs, the node level performance of the current solver is optimized by minimizing memory transfers and using SIMD operations whenever possible, depending on the operation and the type of data [7]. In this

work, the solver is extended to exploit GPUs as co–processors on hybrid machines. The extension is made in a modular way and can be enabled and disabled easily, which allows running the solver efficiently on both CPU-only and hybrid CPU-GPU machines.

As explained in Sec. 5.2, the Poisson solver in the fluid solver is the most expensive component due to the semi-implicit coupling strategy that we follow. Since we have exploited SIMD operations within the solver, GPUs can be used to accelerate these computations. The rest of the computations are performed on the associated CPUs. As explained earlier, we use a PETSc KSP solver to exploit GPUs for solving the pressure Poisson equation in the fluid solver. The PETSc's design separates the application code and the solver and allows using a wide range of GPU programming models such as Kokkos, CUDA, and OpenCL. Since we use NVIDIA GPUs in this work, the CUDA model is selected for the current study. In addition, data are shared between the PETSc programming models and the application code and therefore no explicit copy is needed [121].

An important performance-limiting factor when exploiting GPUs is CPU-GPU communication. The communication must be minimized and preferably overlapped with the computations to minimize the performance loss. To address this issue, PETSc uses pinned memory which allows the RDMA system to use the full bandwidth of the CPU-GPU interconnect for data transfer. In addition, PETSc follows the lazy-mirror model which internally manages two copies of data, one on the CPU and the other one on the GPU. When an operation is done on the GPU, the GPU version of the data is updated, otherwise, the CPU copy of the data is updated and vice-versa. However, when all computations are done on the GPU, there is no copy back data to the CPU. When unified shared memory is available, which is available since CUDA 6 on NVIDIA GPUs, the PETSc back-end class allocates only a single data buffer for both CPUs and GPUs [121]. Therefore, one does not need to explicitly copy data between CPU and GPU and the CUDA library takes care of this communication efficiently.

The data communication among GPUs, both within one node and among multiple nodes, is carried out via the CPUs. GPUs do not communicate directly, for example with using a CUDA-aware MPI. Using pinned memory can facilitate the CPU-GPU data communication, also the inter-GPU communication. Using a CUDA-aware MPI implementation can improve the efficiency by, for example, pipelining the computations and data communication. This is a topic for future works.

In addition, in most exa-scale hybrid machines, the number of CPU cores is higher

than the number of GPUs. Using all CPUs and GPUs will require sharing a single GPU among multiple CPUs, the so-called oversubscription. There is a known overhead cost in current systems for the oversubscription, which is probably due to lunching more kernels with smaller data chunks [121]. Smaller data chunks are not large enough to saturate the GPU capacity and the overall performance degrades. The oversubscription is avoided in the current work by assigning a single GPU to each CPU in the code configuration.

5.4 Inter-Solver Parallelization

This section focuses on the inter-solver parallelization level explained in Sec. 5.3, i.e., the inter-solver communication and load balancing. It is explained how boundary data are exchanged between solvers efficiently. In addition, we explain how inter-solver parallelization can be maintained efficiently and modular in the presence of GPU co-processors. Furthermore, a data-driven load balancing based on the method introduced in Chapter 3 is derived and adapted for an FSI application with a large computational mesh.

5.4.1 Inter-Solver Point-to-Point Communication

Efficient inter-solver communication is a key element in parallel partitioned coupled simulations. This is particularly important in cases with a strong bi-directional coupling between the involved systems, where a high number of coupling iterations, each requiring inter-code communication, is executed to reach a converged solution.

For the current solver, a fully parallel point-to-point communication scheme is incorporated to efficiently exchange boundary data among solvers. Fig. 5.2 schematically shows the different levels of data communication. Data are transferred via CPUs such that there is no direct connection between GPUs. We use TCP/IP sockets to establish the communication channels and perform the necessary data exchange. For this purpose, the fluid and the solid mesh partitions are initially analyzed to find the required communication channels. Here, we use the parallel two-level approach introduced in Chapter 2.

Note that, for inter-solver communication, a separate communicator is built which includes only the interface ranks of both solvers and that is independent of the solvers' `MPI_COMM_WORLD`. Once the communication channels are established, data

are exchanged in an asynchronous way to avoid unnecessary blocking. Establishing the communication channels is carried out only once at the initialization stage. During the rest of the simulation, the same channels are used for data exchange. More details concerning the inter-code data communication can be found in [37, 57].

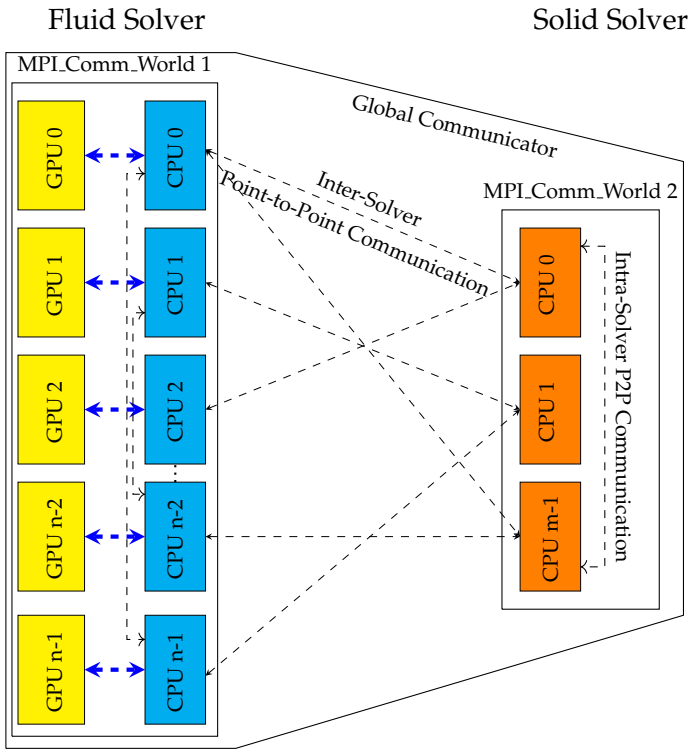


Figure 5.2: Schematic view of the multi-level communication in the coupled fluid-structure framework. The yellow boxes represent the actions on GPUs, the blue ones correspond to fluid solver's CPUs, and the orange boxes to solid solver's CPUs. The gray arrows are used to show inter-CPU communications, both inter-solver, and intra-solver, while the blue arrays represent CPU-GPU communication in the fluid solver.

5.4.2 Inter-Solver Parallelization in the Presence of GPU Co-Processors

We accelerate the fluid solver by using GPU co-processors to improve the overall performance of the coupled solver. The acceleration must make minimum changes to the coupled solver configuration in order to preserve its modularity. We fulfilled this requirement by making the GPU acceleration of the fluid solver transparent to the coupling library and the solid solver. This section describes our setup for the

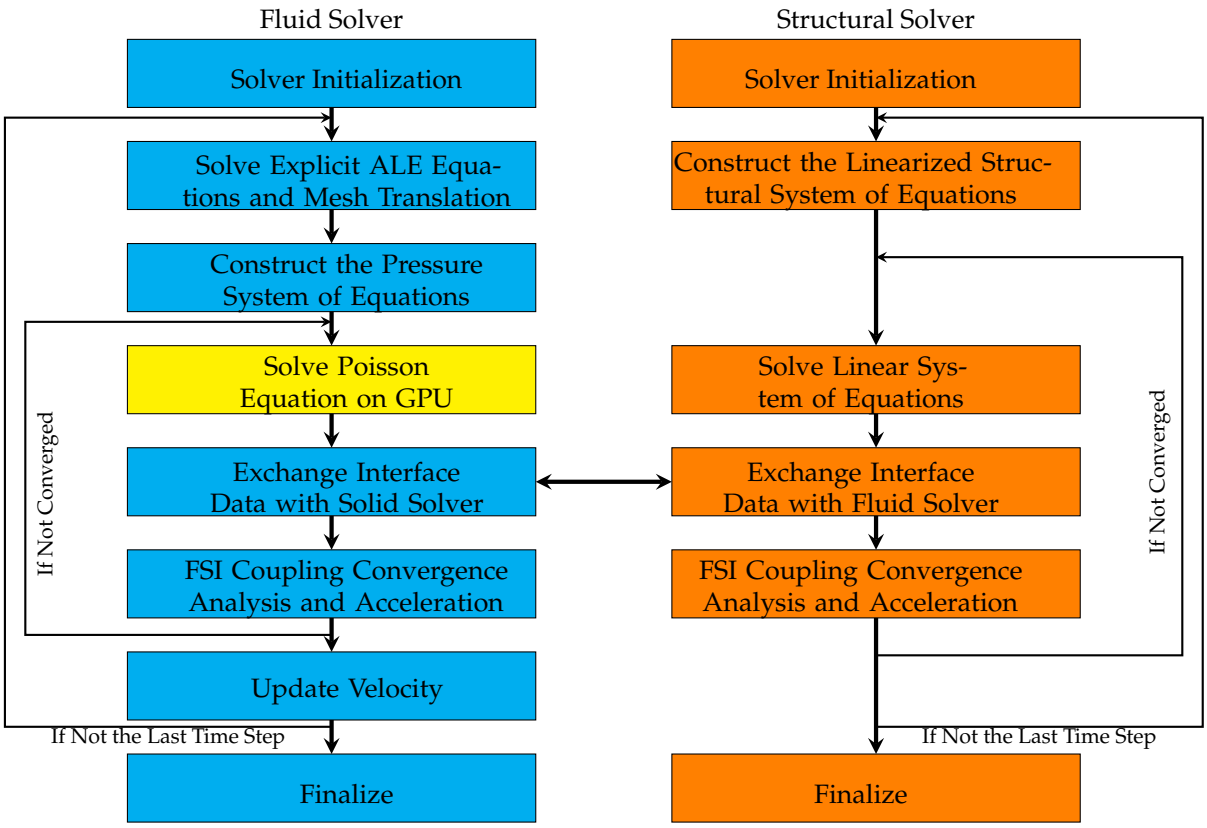


Figure 5.3: Schematic view of the overall algorithm of the coupled FSI solver for exploiting hybrid machines. The yellow box represents the actions on GPUs, the blue ones correspond to the fluid solver's CPUs, and the orange boxes for the solid solver's CPUs.

efficient exploitation of hybrid machines for partitioned FSI simulations. The setup is schematically shown in Fig 5.3. The following steps are taken to efficiently exploit GPUs for FSI simulation:

1. Both solvers initialize on CPUs. This includes creating the required data structures, surface mesh analysis for data mapping, and establishing inter-solver communication channels for boundary data exchange.

2. The fluid solver starts with solving the explicit ALE equations and mesh translation on the CPU, followed by constructing the pressure system of equations to be solved on GPUs. The solid solver constructs the linearized structural system of equations.

3. The solution of the linear systems given by Eq. (5.5) in the fluid solver is accelerated using GPUs. The fluid solver copies the matrix of the pressure equation to the GPU data structures once per time step in a compressed sparse row (CSR)

format. The right-hand side of the liner systems is copied in every coupling iteration. A PETSc KSP solver capable of exploiting GPUs [122, 121] is used to iteratively solve the linear system of equations. The solution vectors are copied back to the CPU to proceed with the remaining steps of the solution algorithm. The structure system of equations is solved on the CPUs.

4. After each iteration, preCICE checks the convergence of the FSI coupling, and data are communicated between the solvers. For this purpose, coupling interface data are exchanged between the single-physics solvers via TCP/IP communication between CPUs. This process is repeated until the FSI coupling convergence is achieved.

5. As soon as the coupling is converged, the final fluid velocity can be calculated using the converged pressure field. This marks the end of the time step and the solver proceeds to the next time step.

5.4.3 Inter-Solver Load Balancing

Load balancing is non-trivial for partitioned FSI simulations. In the current work, inside each single–physics solver, the load is balanced by dividing the computational domain into fairly equal blocks for each process as explained in Sec. 5.3.1. Across the single–physic solvers, we have the condition that both solvers must finish an iteration and exchange the outputs before the next iteration can start. This means, that in case the available computational resources are not distributed optimally among the solvers, one solver will be waiting for the partner. In the current study, we use the multi-step load balancing method that is introduced in Sec. 3.4. For this purpose, (i) we initially build a primary performance model with data gathered from simulations with a small mesh, (ii) scale the models with the ratio between the target mesh size and the small mesh size, and (iii) run only a few short simulations with the target mesh, and finally (iv) establish new performance models using these new data. This method has a lower overhead than a one-step method while it preserves the accuracy of the performance models.

5.5 Performance and Scalability Analysis

To demonstrate the scalability and parallel efficiency of the presented partitioned simulation environment, we present strong scaling measurements for the simulation of blood flow inside a patient-specific aorta. We run scalability tests on a CPU-only and a hybrid CPU-GPU machine, details of which will follow in this section. The measurements are provided on both machines and the results are analyzed. Specifically, (i) we investigate the effect of the data-driven load balancing method on the strong scalability of the CPU-only machine, and (2) we analyze the run time reduction by GPU acceleration.

5.5.1 Test Case Description

The numerical tools to simulate blood flow in the cardiovascular system are constantly developing due to the great clinical interest and due to scientific advances in mathematical models and computational power [123]. The blood flow dynamics investigation can explain many underlying dysfunctions. A better understanding of these dynamics can improve both the disease's diagnosis and the treatment. However, computational analysis of the blood flow inside a patient-specific aorta is very challenging. The research in this field is developing fast concerning both modeling aspects and computational efficiency (e.g. [7, 124, 125]). In the current study, we focus on the computational performance and show that the developed FSI framework is able to efficiently handle such complex simulations on the available hardware.

For the investigations of the current work, the geometry of the aorta provided by the 2nd CFD challenge of the STACOM 2013 conference [126] is used. Figure 5.4 (left) depicts the geometry with inlet and outlet boundaries. Since the thickness of the aortic wall and its mechanical properties were not provided in [126], a value in the typical pathological range $h = 2\mathrm{mm}$ is assumed. For the mechanical properties of the wall, we use the density $\rho_s = 1200\mathrm{kg/m^3}$, the Young modulus $E = 3 \times 10^5 \mathrm{N/m^2}$, and the Poisson ratio $\nu = 0.3$. The density and the viscosity of the blood are assumed to be $\rho_f = 1000\mathrm{kg/m^3}$ and $\mu_f = 0.004\mathrm{Pa \cdot s}$, similar to the values used in [7, 127].

At the inlet, we considered Dirichlet boundary conditions for the fluid velocity, using measured flow rate data provided in [126], along with Neumann boundary conditions for the fluid pressure. At the outlet boundaries, we use explicit RCR Windkessel boundary conditions [128] to model the effect of the rest of the vascular network,

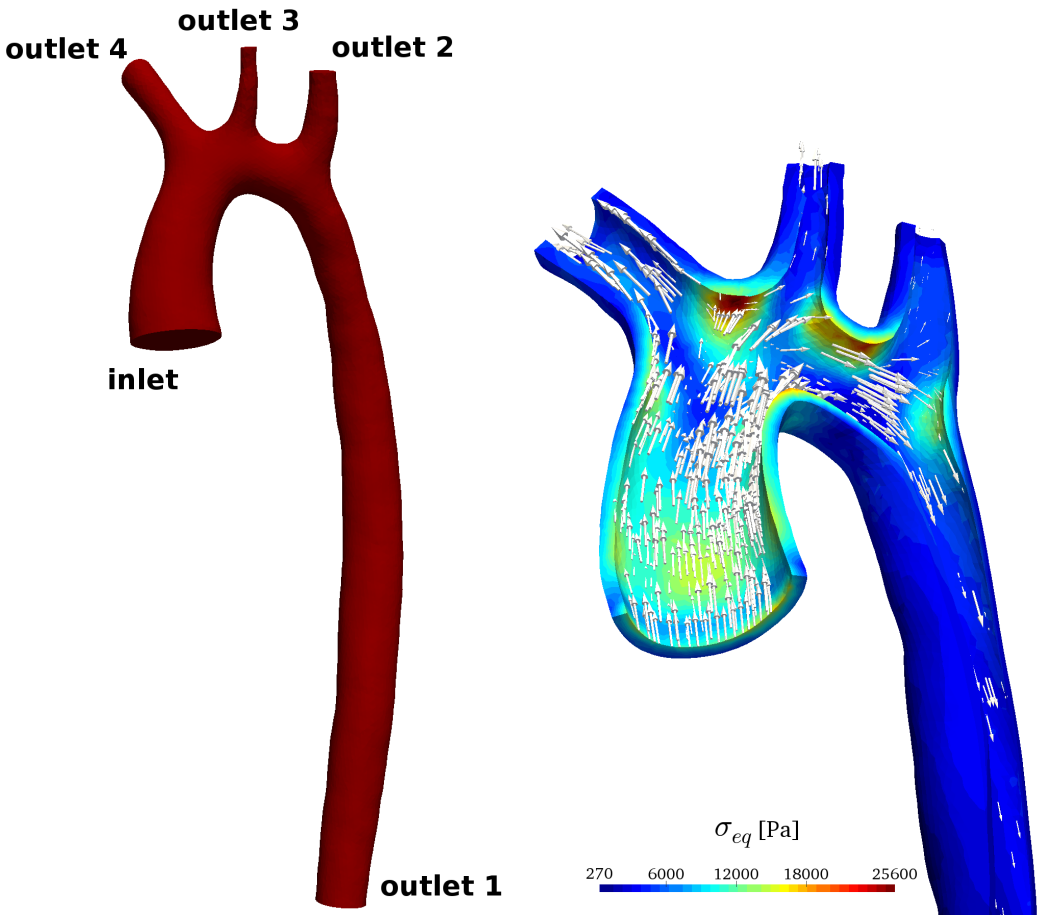


Figure 5.4: Aorta test case: left: 3D geometry of the patient-specific aorta (geometry provided in [126]); right: coupled FSI solution at $t = 0.06$, showing the fluid velocity vectors inside the deformed aortic wall and the von Mises equivalent stress at the wall.

using the Windkessel parameters reported in [129]. For the structure, a clamped boundary condition is used at the inlet and the outlets, while a traction-free boundary condition is assumed on the outer surface of the aortic wall.

For the investigations of the current work, unstructured tetrahedral meshes at three different resolutions are used for the fluid and the solid domain. The mesh information is provided in Table 5.1. The meshes for the fluid and the structure are matching at the interface, thus no extra mapping technique is required. Figure 5.4 (right) shows the coupled FSI solution at a time instant $t = 0.06$s. The figure demonstrates the velocity vectors inside the deformed aortic wall. The color contours in the structural domain correspond to the von Mises equivalent stress.

Table 5.1: Aorta test case: computational grids used for the scalability tests.

Mesh name	No. of cells		
	Fluid	Structure	Common Interface
M1	116M	60M	2M
M2	38M	20M	800K
M3	20M	9M	400K

5.5.2 Hardware Architecture and Numerical Library Description

The scalability measurements on a CPU-only machine are carried out on the SuperMUC NG supercomputer at the Leibniz Supercomputing Centre of the Bavarian Academy of Science and Humanities (LRZ). This machine consists of 3.1GHz Intel Xeon Platinum 8174 (SkyLake) processors. Each node contains two processors with 24 cores per processor (48 cores per node) and 96GB of RAM. The nodes are connected via Intel Omni-Path interconnect. The GNU GCC compiler was used to compile the code. In addition, an Intel MPI implementation compatible with the GCC compiler was used for intra-solver parallelization.

For scalability tests on a hybrid CPU-GPU machine, the Vulcan cluster at the High-Performance Computing Center Stuttgart (HLRS) is used. This machine has 8 heterogeneous compute nodes. Each node consists of 2 $\times$ 2.6GHz Intel Xeon Gold 6240 (CascadeLake) with 36 cores in total and 8 $\times$ NVIDIA Tesla V100 SXM2 GPUs. The total available memory on each node is 768GB. We use the GNU GCC compiler and an Intel MPI implementation compatible with the GCC compiler for intra-solver parallelization and communication.

5.5.3 Strong Scalability on a CPU-Only Machine

In this section, we show the strong scalability of the present partitioned FSI solver for up to 13,440 CPU cores using the computational grid M1 (Table 5.1). The load balancing between the solvers is carried out using the multi-step modeling and optimization approach described in Chapter 3 along with the adjustments given in Sec. 3.4. To build performance models, we first run 5 simulations using mesh M3. We scale the performance models according to mesh sizes to obtain appropriate models for mesh M1 and find the CPU cores distribution. Next, we run 3 more small simulations by using mesh M1 and the initial core distribution found in the first step to obtain the final performance models and load balancing. The core distribution for running the simulations to obtain the performance models are given in Table 5.2. The final core distribution is presented in Table 5.3.

Table 5.2: CPU-only test case: core distribution to build performance models for the first and second step of the multi-step load balancing.

Step	Mesh name	Number of CPU cores		
		Total	Fluid	Structure
1	M3	1440	720	720
1	M3	2400	1200	1200
1	M3	3840	1920	1920
1	M3	4800	2400	2400
1	M3	5760	2880	2880
2	M1	5760	3792	1968
2	M1	9600	5040	4560
2	M1	13440	6240	7200

Figure 5.5 shows the average run time per coupling iteration for different core counts for mesh M1. We observe very good scalability of the solver for up to 13,440 cores, where we achieve a parallel efficiency of 81%. The parallel efficiency for the strong scalability on m cores is evaluated as

$$\text{efficiency} = \frac{f_l \times l}{f_m \times m} \tag{5.17}$$

where f_m is the run time on m cores, and f_l is the run time on the smallest number of cores, indicated by l. If possible, the parallel efficiency is measured against the sequential run time ($l = 1$). However, due to the limitation of memory on a single

Table 5.3: CPU-only test case: run time (per coupling iteration) breakdown into computation and communication times for different number of CPU cores (strong scalability measurements using mesh M1).

total number of CPU cores	number of CPUs fluid	structure	computation time (s) fluid	structure	inter-solver communication time (s)
1920	1200	720	36.47	38.19	3.48
3840	2304	1536	15.86	16.23	2.14
7680	4800	2880	7.29	7.66	1.86
11520	5280	6240	6.47	5.90	1.38
13440	5952	7488	5.95	5.62	1.28
14400	6240	8160	6.62	5.48	1.23

core, large problems cannot be solved sequentially. Therefore, the smallest number of cores that can be used to solve the problem is used as the basis to evaluate the efficiency ($l = 1920$ for this test case).

To analyze efficiency and actual load balancing, we show the scalability graphs for individual solvers and inter-solver communication in Figure 5.6. Moreover, the run time breakdown for different numbers of CPU cores is presented in Table 5.3. We observe very good scalability of the coupled solver for up to 13,440 cores. However, for higher core numbers, the total run time increases and the parallel efficiency degrades. The run time breakdown in Figure 5.6 and Table 5.3 shows, that the performance of the fluid solver degrades for core numbers higher than approximately 6,000, which seems to be due to the mesh size and small arithmetic intensity within the fluid solver's cores. The size of the computational grid (116M for fluid) is not sufficient to be divided efficiently among six thousand processors. The communication time scales up to this point which proves the efficiency of the implemented point-to-point communication scheme. In addition, the run time breakdown shows that there is only a small discrepancy between structure and fluid computation times which proves the effectiveness of the used load balancing scheme.

The performance analysis for the current test case with the single-step load balancing showed strong scalability up to about 10,000 CPU cores in [7]. The multi-step method has improved the scalability by up to approximately 35%. This is because the performance models obtained from the multi-step approach can predict the single-physics solver's scalability limit more accurately. The modification of the initial performance models by a few measurements from the target mesh increases the accuracy of the nonlinear part of the performance models (where the scalability starts

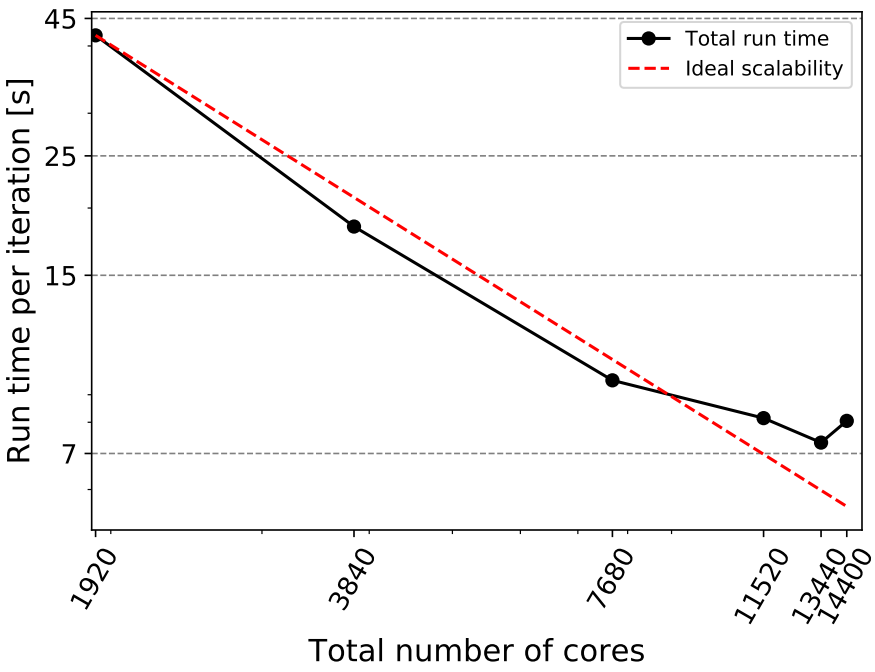


Figure 5.5: CPU-only test case: average (per iteration per core) run time for different core counts (strong scalability test results using mesh M1).

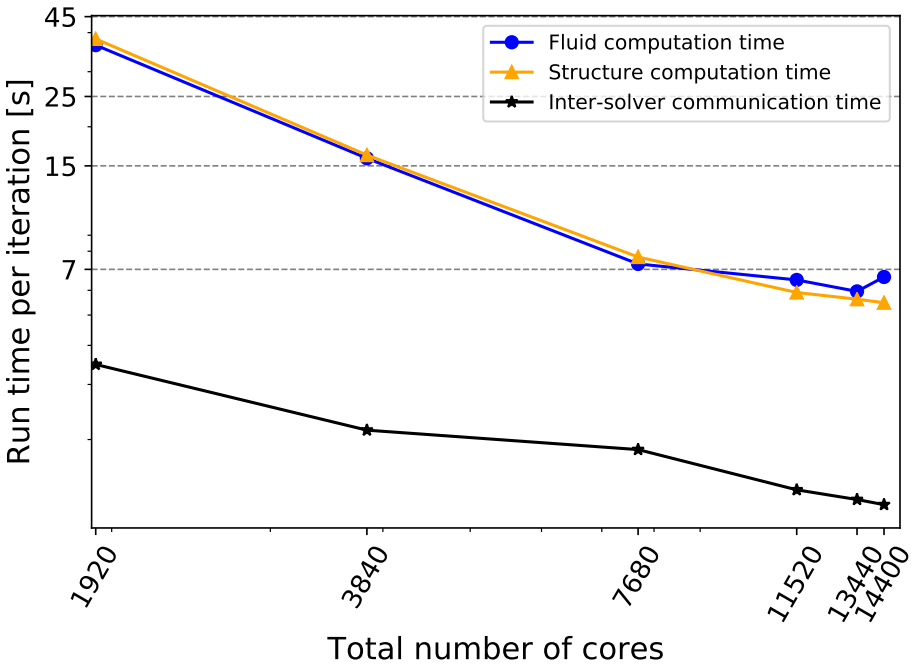


Figure 5.6: CPU-only test case: average run time breakdown (per iteration per core) for different core counts (strong scalability test results using mesh M1).

to degrade). For the current test case, the multi-step method can predict the fluid solver's scalability loss, while the single-step method is not capable of such a prediction. We compare the parallel scalability of the current solver to similar FSI solvers in literature in Table 5.4. The cited works follow a partitioned approach to solve large-scale FSI problems. Other examples in literature, which did not present scalability measurements, are not cited here. Despite many advantages that the partitioned methods offer, loss of parallel efficiency in multi-code coupling is considered one of their main drawbacks. This is due to the challenges in terms of data structuring, equation coupling, domain decomposition, parallel data communication, and inter-solver load balancing. By addressing these issues in the current work, our coupled solver is shown to scale up to 13,440 cores with a very high parallel efficiency.

Table 5.4: CPU-only test case: Scalability of the presented simulation environment compared to other partitioned FSI solvers in literature.

Source	Spatial grid type	Scalability (CPU cores)	Parallel efficiency on highest core count (%)
Present work	Unstructured	13,440	81
Naseri et al. (2020) [7]	Unstructured	10,080	83
Hewitt et al. (2019) [110]	Structured	1,536	62
Cajas et al. (2018) [29]	Unstructured	768	68

5.5.4 Strong Scalability on a Hybrid CPU-GPU Machine

To demonstrate the potential for acceleration of the coupled simulation by using GPUs in addition to CPUs, strong scalability measurements are presented for the Aorta test case on a hybrid CPU-GPU machine.

The provided analysis includes the GPU acceleration effect on both the performance of the fluid solver within the coupled FSI solver and on the overall performance of the coupled solver. We run coupled simulations using unstructured tetrahedral meshes presented in Table 5.1 with different numbers of compute nodes. In order to show the GPU acceleration effect, the coupled simulations are carried out twice: once with GPU acceleration of the fluid solver and once without using any GPUs. For hybrid CPU-GPU simulations, the fluid solver uses an equal number of CPUs and GPUs, i.e., from each node 8 GPUs and 8 CPU cores, to avoid oversubscription

(see Sec. 5.3.2). On the other hand, for the fluid solver in CPU-only simulations and the solid solver in both scenarios, all of the 36 available CPU cores within each node are exploited. Run times for both scenarios and the respective speed-up due to the GPU acceleration are presented in Table 5.5 and 5.6. Table 5.5 shows the performance data of the fluid solver during the coupled simulations. The measurements show, that by accelerating the fluid solver, we get up to 133.4 times speed-up. We observe a decline in the speed-up when increasing the number of GPUs. This is probably due to the smaller data chunk when the mesh is decomposed into smaller partitions that is not enough to saturate the computation capacity of the GPU.

Furthermore, to investigate the GPU-acceleration effect on the overall performance of the coupled solver, we compare the coupled simulation run times with and without GPU acceleration for mesh M2. This comparison is carried out for a total number of 8 compute nodes (machine capacity). The information about node distribution and run times is provided in Table 5.6. A 7.2 times faster simulation is achieved by accelerating the fluid solver using GPUs. Note that the overall speed-up is currently limited by the available total number of compute nodes. In a larger machine, where a larger mesh can be used along with a finer compute node distribution, a higher speed-up can be achieved. The current setup does not allow the desired distribution of computational resources, since only complete nodes can be assigned to solvers. For instance, if the optimal number of CPU cores for the solid solver is 54, we either have to use 36 cores (1 node) or 72 cores (2 nodes). This degrades the load balancing and deteriorates the overall performance.

The strong scalability measurements are presented to analyze the overall perfor-

Table 5.5: Hybrid CPU-GPU test case: fluid solver speed up by GPU acceleration (run times measured in a coupled simulation).

Mesh	number of			computation time (s)		speed-up
	nodes	CPUs	GPUs	with GPU	without GPU	
M1	2	16	16	22.9	2735.2	119.6
M1	4	32	32	16.9	1367.6	80.7
M2	1	8	8	15.8	2128.3	133.4
M2	2	16	16	12.1	1064.1	88.2
M3	1	8	8	8.3	992.3	119.0
M3	2	16	16	7.1	495.2	69.4

mance of the framework on a hybrid CPU-GPU machine. For these simulations, the mesh M2 is used. The node distribution are presented in Table 5.7 for the hybrid simulations. The computational resources distribution is carried out using node counts

Table 5.6: Hybrid CPU-GPU test case: coupled solver's speed up by GPU acceleration (coupled simulation using mesh M2).

	CPU+GPU		CPU-only		run time (s)		speed-up
	fluid	structure	fluid	structure	CPU+GPU	CPU-only	
nodes	2	6	6	2			
CPUs	16	216	216	72	28.2	201.1	7.2
GPUs	16	0	-	-			

instead of core counts in order to avoid node sharing between solvers. Figure 5.7 depicts the average run time per iteration for different numbers of compute nodes. The number of nodes equals the total number of nodes exploited by the fluid and the solid solver together. A very good reduction in computational time is achieved by increasing the number of nodes up to 8 nodes, which is the full machine capacity, with a parallel efficiency of 94% on 8 nodes. The parallel efficiency is defined in Eq. 5.17.

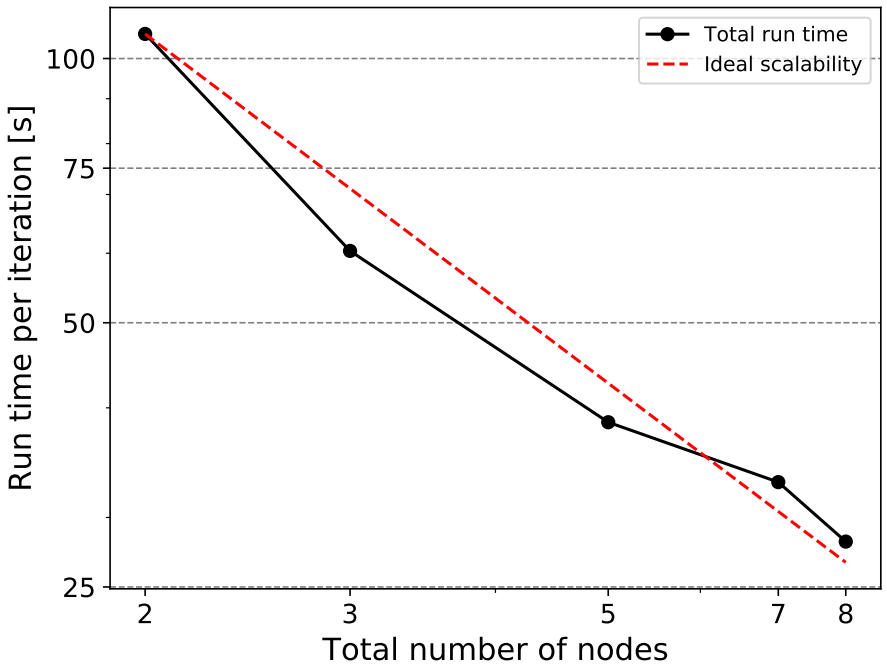


Figure 5.7: Hybrid CPU-GPU test case: average run time (per iteration per core) for different core counts (strong scalability test results using mesh M2). For the fluid solver, an equal number of CPUs and GPUs (8 per node) is exploited. While, for the solid solver, all available CPUs (36) within each node are used.

To further study the efficiency and actual load balancing, the run time breakdown is presented for different numbers of nodes in Fig. 5.8 and Table 5.7. The GPU acceleration improvement on the fluid solver is very significant and thus, the fluid solver is much faster than the solid solver. Due to the file system limitation, only complete nodes can be allocated to each solver. Therefore, the minimum number of GPUs that can be exploited by the fluid solver is 8. It is observed that the solid solver's computation time is much higher than the accelerated fluid solver's run time if an equal number of computational resources is used for both solvers. Therefore, to minimize the computation time gap between solvers, more and more nodes must be allocated to the solid solver, while preserving the fluid solver's nodes. Due to the constraint of allocating complete nodes to solvers and the limitation of the computational resources, we are not able to use the introduced data-driven load balancing method for the hybrid CPU-GPU simulations. On a larger machine, where a higher number of nodes is available for the solid solver, or in a case where the file system allows node sharing between solvers, much better load balancing can be achieved by incorporating our load balancing method.

Furthermore, the point-to-point inter-solver communication time scales by the number of nodes. The inter-solver communication's contribution to the total run time is significant. This is because the GPU acceleration has significantly reduced the compute time even when using a few nodes. However, since the communication is performed via CPUs, it directly depends on the number of CPU cores. Accordingly, using a low number of nodes results in a significant communication time as each node needs to exchange a large amount of interface data (data chunks are larger (per core) due to the smaller number of partitions compared to the CPU-only scenario).

Table 5.7: Hybrid CPU-GPU test case: run time breakdown into computation and communication times for different number of nodes (strong scalability measurements using mesh M2).

fluid			solid		computation time (s)		inter-solver
nodes	CPUs	GPUs	nodes	CPUs	fluid	structure	comm. time (s)
1	8	8	1	36	15.72	58.10	81.65
1	8	8	2	72	15.76	29.74	34.81
1	8	8	4	144	15.79	15.31	22.98
1	8	8	6	216	15.77	11.32	21.39
2	16	16	6	216	12.06	11.21	20.76

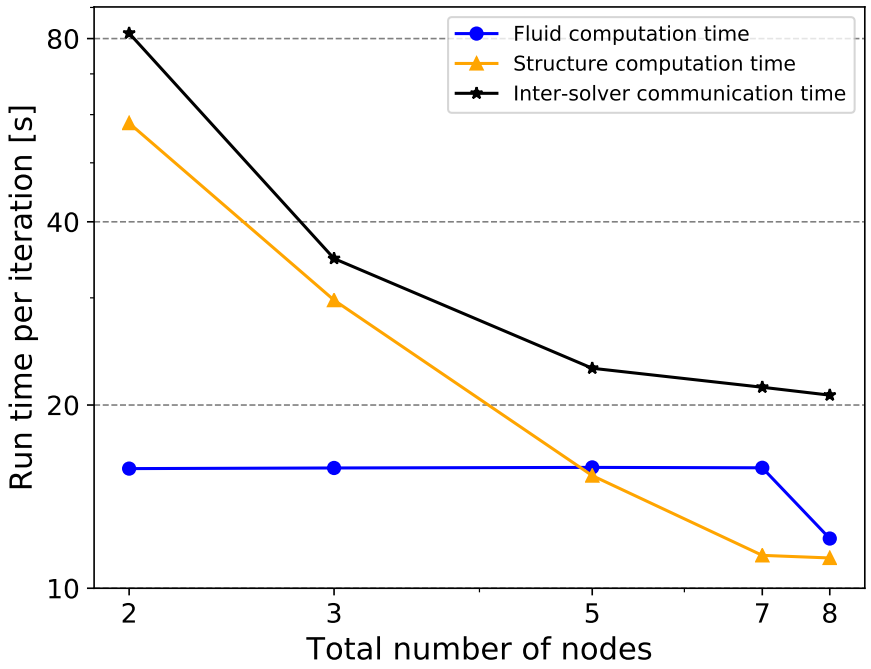


Figure 5.8: Hybrid CPU-GPU test case: average run time breakdown (per iteration per core) for different core counts (strong scalability test results using mesh M2).

5.6 Summary

We presented a strongly-coupled partitioned fluid-structure interaction simulation environment for massively parallel CPU architectures and hybrid CPU-GPU machines. The partitioned approach is generally known to be highly flexible in terms of the coupled codes and even the coupled types of equations, but also to be highly challenging in terms of an efficient implementation on parallel architectures. The key to success in our work was to use single–physic solvers and a coupling library that were both already highly scalable on CPU machines up to very high core numbers and to combine them with our new contributions. In terms of coupling, parallel efficiency is already a feature that is offered only by very few libraries without introducing central communication instances that deteriorate scalability. We enhanced this setup with three main new aspects: a semi-implicit coupling strategy that segregates the fluid pressure terms and strongly couples it to the solid solver (Naseri et. al [7]), a novel inter-code load-balancing and a generalizable approach to exploit hybrid CPU-GPU architectures.

For inter-solver load balancing, we applied a data-driven multi-step approach. The important innovative aspects are (i) the data-driven performance modeling allowing us to overcome limitations of analytical approaches in the presence of different types of equations and discretization and iterative solver usage; (ii) the multi-step approach that allows us to establish the required performance models with very low computational cost by using many data points for a coarse simulation in a first modeling step followed by a second step, where fine simulation data are generated already close to the final regime and, thus, very few data points are sufficient.

To port the whole coupled simulation environment to hybrid CPU-GPU architectures, the coupled solver is analyzed to identify the computationally most expensive parts, in our application the solver for the fluid pressure equation. These components are offloaded to GPUs, which happens completely transparent to the coupling library such that it does not interfere with the parallel efficiency of the inter-code communication and coupling numerics.

To show the efficiency of the developed framework, a strong scalability and performance analysis is carried out for a patient-specific aorta test case on both CPU-only hardware and a hybrid CPU-GPU machine. CPU-only scalability measurements using a mesh with 116 million cells for the fluid and 60 million cells for the structure solver showed a parallel efficiency of 81% on 13,440 CPU cores. Investigations on the hybrid CPU-GPU machine for a mesh with 38 million cells for the fluid and

20 million cells for structure demonstrated an excellent run time reduction for the fluid solver by a factor of 133 and for the coupled solver by a factor of 7.2 compared to CPU-only simulation. In addition, we observed almost ideal scalability for the coupled solver on up to 8 nodes with a parallel efficiency of 94%.

6 Summary and Conclusion

This thesis focused on the performance improvement of partitioned multi-physics simulations by using data-integrated methods. The main aim of this work was to prepare coupled solvers for efficient execution on large supercomputers. Following a partitioned approach, where the simulation domain is decomposed into smaller subdomains and each is solved by a separate solver, allows using (i) available single-physics solvers and (ii) well-validated numerical methods. Consequently, the partitioned approach reduces the software development cost and enhances flexibility. However, it introduces new challenges, in particular when the parallel performance of the simulation is considered. These challenges include, but are not limited to, inter-solver data communication, load balancing, and equation coupling. Each chapter of this thesis addressed one of these issues.

Chapter 2 improved the inter-solver communication initialization of the coupling library preCICE. The enhancement is achieved by replacing the previous one-level method with an efficient two-level approach. The new scheme improved the scalability of the initialization by replacing gather-scatter mesh communication with a parallel point-to-point scheme. In addition, the memory bottleneck of the old method, which was due to gathering the complete interface mesh in a master rank, is completely removed. The numerical performance analysis showed, that the two-level scheme is up to five times faster compared to the previous one. In addition, by resolving the memory issues, preCICE can now handle the coupling of much larger computational meshes.

The inter-solver load balancing for highly parallel multi-physics simulations is investigated in Chapter 3. We introduced a data-driven approach to efficiently distribute the available computational resources among coupled single-physics solvers. Two different methods were considered to model the performance of single-physics solvers – PMNF regression and deep neural networks. The former is capable of establishing accurate single-variable (number of processors) models, while, the latter can be used for multi-variable (number of processors and problem size) modeling. The multi-variable scheme can build accurate performance models using the performance data gathered from smaller meshes. This significantly reduces the load balancing overhead. The performance models are then incorporated into an integer optimization problem to calculate the optimal computational resources distribution

among solvers. The performance investigation indicates, that the proposed method can almost completely remove the load imbalance between solvers. Doing so significantly improves the simulation performance and scalability. The presented method is, however, only suitable for solvers without dynamic re-meshing. For this case, the method can be extended to adapt the inter-solver load balancing, if any of the single-physics solvers updates its mesh partitioning.

Chapter 4 introduced a hybrid data-integrated approach to accelerate the solution convergence in strongly coupled simulations. A deep neural network is trained to estimate the solution for each time step, which can be used as a first guess by the numerical solvers to compute the final solution. An on-the-fly re-training strategy is also presented to avoid the so-called model-data inconsistency. The presented scheme updates the DNN during the simulation to preserve the estimation accuracy. For a one-dimensional FSI test case, the presented acceleration scheme with the on-the-fly re-training can reduce the number of classical iterations to only three. The method significantly improves even sophisticated acceleration schemes such as quasi-Newton, which require at least six iterations for each time step. In addition, a training and inference parallelization scheme for DNNs based on the data decomposition is introduced in Chapter 4. The introduced approach is compatible with the method used in numerical simulation codes. The presented technique can be incorporated to integrate the DNN surrogate solvers into parallel simulation codes. To use the DNN acceleration for more complex test cases, such as those with unstructured meshes, more steps are necessary. For instance, graph-based neural networks with the capability of applying convolution can be used for test cases with unstructured meshes. In addition, the acceleration must be further investigated to study the effect of DNN acceleration on numerical methods, such as quasi-Newton, for the problems with two or three-dimensional meshes.

Finally, in Chapter 5, a strongly-coupled partitioned fluid-structure interaction simulation environment is introduced for massively parallel CPU architectures and hybrid CPU-GPU machines. The simulation environment coupled highly scalable single-physics solvers using the coupling library preCICE for an efficient FSI simulation. In addition, the coupled framework is extended to exploit multi-GPU machines by offloading the most compute-intensive kernels of the simulation to GPU cores. To maximize the computational efficiency of this framework, we employed the data-driven multi-step load balancing scheme introduced in Chapter 3 for simulations on CPU-only machines. The strong scalability and performance analysis on a CPU machine showed a parallel efficiency of 81% on 13,440 CPU cores. In addition, com-

putational performance analysis on a hybrid CPU-GPU machine demonstrated a significant run time reduction for the fluid solver by a factor of 133 and the coupled solver by a factor of 7.2 compared to CPU-only simulations with a parallel efficiency of 94%.

Bibliography

[1] Jed Brown et al. "Composable Linear Solvers for Multiphysics." In: *2012 11th International Symposium on Parallel and Distributed Computing*. 2012, pp. 55–62. DOI: 10.1109/ISPDC.2012.16.

[2] Simense Simulation Center. *Multiphysics Simulation, Accurately predict physical product performance*. http://siemens-simcenter.tridiagonal.com/simcenter-3d/multiphysics-simulation/. [Online; accessed 26-01-2022].

[3] Hans-Joachim Bungartz and Michael Schäfer. *Fluid-structure interaction: modelling, simulation, optimisation*. Vol. 53. Springer Science & Business Media, 2006.

[4] Boyce E Griffith and Neelesh A Patankar. "Immersed methods for fluid–structure interaction." In: *Annual review of fluid mechanics* 52 (2020), pp. 421–448.

[5] Michael Neunteufel and Joachim Schöberl. "Fluid-structure interaction with H (div)-conforming finite elements." In: *Computers & Structures* 243 (2021), p. 106402.

[6] Rajeev Kumar Jaiman and Vaibhav Joshi. *Computational Mechanics of Fluid-Structure Interaction*. 2022.

[7] Alireza Naseri et al. "A scalable framework for the partitioned solution of fluid–structure interaction problems." In: *Computational Mechanics* 66 (2020), pp. 471–489.

[8] Weili Jiang, Xudong Zheng, and Qian Xue. "Computational modeling of fluid–structure–acoustics interaction during voice production." In: *Frontiers in bioengineering and biotechnology* 5 (2017), p. 7.

[9] Li Wang and Fang-Bao Tian. "Numerical study of flexible flapping wings with an immersed boundary method: Fluid–structure–acoustics interaction." In: *Journal of Fluids and Structures* 90 (2019), pp. 396–409.

[10] Neda Ebrahimi Pour et al. "Coupled simulation with two coupling approaches on parallel systems." In: *Sustained Simulation Performance 2017*. Springer, 2017, pp. 151–164.

[11] Florian Lindner et al. "ExaFSA: Parallel Fluid-Structure-Acoustic Simulation." In: *Software for Exascale Computing-SPPEXA 2016-2019*. Springer, 2020, pp. 271–300. DOI: 10.1007/978-3-030-47956-5_10.

[12] L He and MLG Oldfield. "Unsteady conjugate heat transfer modeling." In: (2011).

[13] Jinku Wang, Moran Wang, and Zhixin Li. "A lattice Boltzmann algorithm for fluid–solid conjugate heat transfer." In: *International journal of thermal sciences* 46.3 (2007), pp. 228–234.

[14] Bibin John, P Senthilkumar, and Sreeja Sadasivan. "Applied and theoretical aspects of conjugate heat transfer analysis: a review." In: *Archives of Computational Methods in Engineering* 26.2 (2019), pp. 475–489.

[15] Y. Bazilevs et al. "Isogeometric fluid-structure interaction analysis with applications to arterial blood flow." In: *Computational Mechanics* 38.4-5 (2006), pp. 310–322. DOI: 10.1007/s00466-006-0084-3.

[16] Kenji Takizawa, Yuri Bazilevs, and Tayfun E Tezduyar. "Space–time and ALE–VMS techniques for patient-specific cardiovascular fluid–structure interaction modeling." In: *Archives of Computational Methods in Engineering* 19.2 (2012), pp. 171–225. DOI: 10.1007/s11831-012-9071-3.

[17] Philip Cardiff and Ismet Demirdžić. "Thirty years of the finite volume method for solid mechanics." In: *arXiv preprint arXiv:1810.02105* (2018).

[18] Joris Degroote. "Partitioned simulation of fluid-structure interaction." In: *Archives of Computational Methods in Engineering* 20 (2013), pp. 185–238. ISSN: 1134-3060. DOI: 10.1007/s11831-013-9085-5.

[19] Gene Hou, Jin Wang, and Anita Layton. "Numerical methods for fluid-structure interaction - A review." In: *Communications in Computational Physics* 12.2 (2012), pp. 337–377. ISSN: 18152406. DOI: 10.4208/cicp.291210.290411s.

[20] Amin Totounferoush et al. "A GPU Accelerated Framework for Partitioned Solution of Fluid-Structure Interaction Problems." In: *14th WCCM-ECCOMAS Congress 2020*. Vol. 700. 2021.

[21] Miriam Mehl et al. "Parallel coupling numerics for partitioned fluid–structure interaction simulations." In: *Computers & Mathematics with Applications* 71.4 (2016), pp. 869–891. DOI: 10.1016/j.camwa.2015.12.025.

[22] Henry G Weller et al. "A tensorial approach to computational continuum mechanics using object-oriented techniques." In: *Computers in physics* 12.6 (1998), pp. 620–631.

[23] Guido Dhondt and Klaus Witting. *CalculiX, A Free Software Three-Dimensional Structural Finite Element Program*.

[24] Martin Alnæs et al. "The FEniCS project version 1.5." In: *Archive of Numerical Software* 3.100 (2015).

[25] *ANSYS Fluent User's Guide, 2019R1*. ansys.com. 2019.

[26] O Lehmkuhl et al. "TermoFluids: A new Parallel unstructured CFD code for the simulation of turbulent industrial problems on low cost PC Cluster." In: *Parallel Computational Fluid Dynamics 2007*. Springer, 2009, pp. 275–282. DOI: 10.1007/978-3-540-92744-0-34.

[27] *TermoFluids CFD software*. www.termofluids.com. 2019.

[28] U. Küttler and W. A. Wall. "Fixed-point fluid-structure interaction solvers with dynamic relaxation." In: *Computational Mechanics* 43 (2008), pp. 61–72. DOI: 10.1007/s00466-008-0255-5.

[29] Juan C Cajas et al. "Fluid-structure interaction based on HPC multicode coupling." In: *SIAM Journal on Scientific Computing* 40.6 (2018), pp. C677–C703.

[30] Jean Frédéric Gerbeau and Marina Vidrascu. "A quasi-Newton algorithm based on a reduced model for fluid-structure interaction problems in blood flows." In: *ESAIM: Mathematical Modelling and Numerical Analysis* 37 (2003), pp. 631–647. DOI: 10.1051/m2an:2003049.

[31] C Michler, E H Van Brummelen, and R De Borst. "An interface Newton-Krylov solver for fluid-structure interaction." In: *International Journal for Numerical Methods in Fluids* 47.10-11 (2005), pp. 1189–1195. DOI: 10.1002/fld.850.

[32] Klaudius Scheufele and Miriam Mehl. "Robust Multisecant Quasi-Newton Variants for Parallel Fluid-Structure Simulations–and Other Multiphysics Applications." In: *SIAM Journal on Scientific Computing* 39.5 (2017), S404–S433. DOI: 10.1137/16M1082020.

[33] Hans-Joachim Bungartz et al. "A plug-and-play coupling approach for parallel multi-field simulations." In: *Computational Mechanics* 55.6 (2015), pp. 1119–1129. DOI: 10.1007/s00466-014-1113-2.

[34] Gerasimos Chourdakis et al. "PRECICE V2: ASustainable AND USER-FRIENDLY COUPLING LIBRARY." In: *arXiv preprint arXiv:2109.14470* (2021).

[35] Joris Degroote et al. "Stability of a coupling technique for partitioned solvers in FSI applications." In: *Computers & Structures* 86.23-24 (2008), pp. 2224–2234.

[36] Jan Vierendeels et al. "Stability issues in partitioned FSI calculations." In: *Fluid Structure Interaction II*. Springer, 2011, pp. 83–102.

[37] Hans-Joachim Bungartz et al. "preCICE–a fully parallel library for multi-physics surface coupling." In: *Computers & Fluids* 141 (2016), pp. 250–258. DOI: 10.1016/j.compfluid.2016.04.003.

[38] R Haelterman et al. "Improving the performance of the partitioned QN-ILS procedure for fluid–structure interaction problems: Filtering." In: *Computers & Structures* 171 (2016), pp. 9–17. DOI: 10.1016/j.compstruc.2016.04.001.

[39] Alexander K Shukaev. "A fully parallel process-to-process intercommunication technique for precice." In: *Master's thesis, Institut für Informatik, Technische Universität München* (2015).

[40] Wolfgang Joppich and M Kürschner. "MpCCI—a tool for the simulation of coupled applications." In: *Concurrency and computation: Practice and Experience* 18.2 (2006), pp. 183–192. DOI: 10.1002/cpe.913.

[41] Florent Duchaine et al. "Analysis of high performance conjugate heat transfer with the openpalm coupler." In: *Computational Science & Discovery* 8.1 (2015), p. 015003. DOI: 10.1088/1749-4699/8/1/015003.

[42] T Wang et al. "Concept and realization of the coupling software EMPIRE in multiphysics co-simulation." In: *MARINE V: proceedings of the V International Conference on Computational Methods in Marine Engineering*. CIMNE. 2013, pp. 289–298.

[43] Sophie Valcke, Tony Craig, and Laure Coquart. "OASIS3-MCT user guide, oasis3-mct 2.0." In: *CERFACS/CNRS SUC URA* 1875 (2012).

[44] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.0*. June 2021. URL: https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf.

[45] Aukje de Boer, Alexander H van Zuijlen, and Hester Bijl. "Review of coupling methods for non-matching meshes." In: *Computer methods in applied mechanics and engineering* 196.8 (2007), pp. 1515–1525.

[46] Aukje de Boer, Alexander H van Zuijlen, and Hester Bijl. "Comparison of conservative and consistent approaches for the coupling of non-matching meshes." In: *Computer Methods in Applied Mechanics and Engineering* 197.49-50 (2008), pp. 4284–4297.

[47] N Maman and Charbel Farhat. "Matching fluid and structure meshes for aeroelastic computations: a parallel approach." In: *Computers & Structures* 54.4 (1995), pp. 779–785.

[48] Juan Raul Cebral and Rainald Lohner. "Conservative load projection and tracking for fluid-structure problems." In: *AIAA journal* 35.4 (1997), pp. 687–692.

[49] Thomas CS Rendall and Christian B Allen. "Unified fluid–structure interpolation and mesh motion using radial basis functions." In: *International journal for numerical methods in engineering* 74.10 (2008), pp. 1519–1559.

[50] Gregory E Fasshauer and Michael J McCourt. *Kernel-based approximation methods using Matlab*. Vol. 19. World Scientific Publishing Company, 2015.

[51] Florian Lindner. "Data transfer in partitioned multi-physics simulations: interpolation and communication." PhD thesis. University of Stuttgart, 2019. DOI: 10.18419/opus-10581.

[52] Matthias Lieber and Wolfgang E Nagel. "Highly scalable sfc-based dynamic load balancing and its application to atmospheric modeling." In: *Future Generation Computer Systems* 82 (2018), pp. 575–590.

[53] Martin Schreiber and Hans-Joachim Bungartz. "Cluster-based communication and load balancing for simulations on dynamically adaptive grids." In: *Procedia Computer Science* 29 (2014), pp. 2241–2253.

[54] Benjamin Uekermann. "Partitioned fluid-structure interaction on massively parallel systems." PhD thesis. Munich, Germany: Department of Informatics, Technical University of Munich, 2016. DOI: 10.14459/2016md1320661.

[55] Amin Totounferoush et al. "Parallel Machine Learning of Partial Differential Equations." In: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2021, pp. 698–703.

[56] Hans-Joachim Bungartz et al. "Partitioned fluid-structure-acoustics interaction on distributed data – coupling via preCICE." In: *Software for Exa-scale Computing – SPPEXA 2013-2015*. Ed. by Hans-Joachim Bungartz, Philipp Neumann, and E. Wolfgang Nagel. Springer, 2016. DOI: 10.1007/978-3-319-40528-5_11.

[57] Amin Totounferoush et al. "Efficient and Scalable Initialization of Partitioned Coupled Simulations with preCICE." In: *Algorithms* 14.6 (2021), p. 166. DOI: 10.3390/a14060166.

[58] Klaus Wolf et al. "MpCCI: neutral interfaces for multiphysics simulations." In: *Scientific Computing and Algorithms in Industrial Simulations*. Springer, 2017, pp. 135–151. DOI: 10.1007/978-3-319-62458-7_7.

[59] S Slattery, P Wilson, and R Pawlowski. "The data transfer kit: a geometric rendezvous-based tool for multiphysics data transfer." In: *International conference on mathematics & computational methods applied to nuclear science & engineering (M&C 2013)*. 2013, pp. 5–9.

[60] Steven J Plimpton, Bruce Hendrickson, and James R Stewart. "A parallel rendezvous algorithm for interpolation between multiple grids." In: *Journal of Parallel and Distributed Computing* 64.2 (2004), pp. 266–276. DOI: 10.1016/j.jpdc.2003.11.006.

[61] Yu-Hang Tang et al. "Multiscale universal interface: a concurrent framework for coupling heterogeneous solvers." In: *Journal of Computational Physics* 297 (2015), pp. 13–31. DOI: 10.1016/j.jcp.2015.05.004.

[62] David Thomas et al. "CUPyDO-An integrated Python environment for coupled fluid-structure simulations." In: *Advances in Engineering Software* 128 (2019), pp. 69–85. DOI: 10.1016/j.advengsoft.2018.05.007.

[63] Satish Balay et al. *PETSc Web page*. https://petsc.org/. 2021. URL: https://petsc.org/.

[64] Christophe Geuzaine and Jean-François Remacle. "Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities." In: *International journal for numerical methods in engineering* 79.11 (2009), pp. 1309–1331. DOI: 10.1002/nme.2579.

[65] S. Roller et al. "An Adaptable Simulation Framework Based on a Linearized Octree." In: *High Performance Computing on Vector Systems 2011*. Ed. by M. Resch et al. Springer Berlin Heidelberg, 2012, pp. 93–105. ISBN: 978-3-642-22244-3.

[66] Alexandru Calotoiu et al. "Fast multi-parameter performance modeling." In: *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2016, pp. 172–181.

[67] Amin Totounferoush et al. "A new load balancing approach for coupled multiphysics simulations." In: *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2019, pp. 676–682.

[68] Amin Totounferoush et al. "A data-based inter-code load balancing method for partitioned solvers." In: *Journal of Computational Science* 51 (2021), p. 101329.

[69] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[70] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[71] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks." In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[72] Juri Leonhard Schröder. "Load-balancing for multi-physics simulations." MA thesis. Universität Stuttgart, 2019.

[73] Diederik Kingma and Jimmy Ba. "Adam: a method for stochastic optimization (2014)." In: *arXiv preprint arXiv:1412.6980* 15 (2015).

[74] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[75] Francois Chollet. *Deep Learning with Python and Keras: The Handbook by the Developer of the Keras Library*. MITP-Verlag GmbH & Co. KG, 2018.

[76] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[77] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Searching for activation functions (2017)." In: *arXiv preprint arXiv:1710.05941* (2017).

[78] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[79] Ning Qian. "On the momentum term in gradient descent learning algorithms." In: *Neural networks* 12.1 (1999), pp. 145–151.

[80] Jiaxing Qi. "Efficient Lattice Boltzmann Simulations on Large Scale High Performance Computing Systems." PhD thesis. RWTH Aachen University, 2017.

[81] Daniel F. Harlacher et al. "Dynamic Load Balancing for Unstructured Meshes on Space-Filling Curves." In: *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*. May 2012, pp. 1661–1669.

[82] Harald G. Klimach et al. "Distributed Octree Mesh Infrastructure for Flow Simulations." In: *ECCOMAS 2012 - European Congress on Computational Methods in Applied Sciences and Engineering, e-Book Full Papers*. Ed. by J Eberhardsteiner. 2012.

[83] Neda Ebrahimi Pour et al. "Load Balancing for Immersed Boundaries in Coupled Simulations." In: *Sustained Simulation Performance 2018 and 2019*. Ed. by Michael M. Resch et al. Springer International Publishing, 2019, pp. 185–201. DOI: 10.1007/978-3-030-39181-2_15.

[84] Simulationstechnik und Wissenschaftliches Rechnen Uni Siegen. *Ateles Source Code*. https://osdn.net/projects/apes/scm/hg/ateles/. [Online; accessed 26-08-2019]. 2019.

[85] Neda Ebrahimi Pour and Sabine Roller. "Error investigation for coupled simulations using Discontinuous Galerkin method for discretisation." In: *In Proceedings of ECCM VI / ECFD VII*. Glasgow, UK, June 2018.

[86] Nikhil Anand et al. "Utilization of the Brinkman Penalization to Represent Geometries in a High-Order Discontinuous Galerkin Scheme on Octree Meshes." In: *Symmetry* 11.9 (2019), pp. 11–26. DOI: 10.3390/sym11091126.

[87] Neda Ebrahimi Pour et al. "Coupled Simulation with Two Coupling Approaches on Parallel Systems." In: *Sustained Simulation Performance 2017*. Ed. by Michael M. Resch et al. Springer International Publishing, 2017, pp. 151–164. DOI: 10.1007/978-3-319-66896-3_10.

[88] Harald Klimach and Neda Ebrahimi Pour. *Investigating Coupled Fluid-Structure-Acoustic (FSA) Interaction*. https://www.gauss-centre.eu/results/computational-and-scientific-engineering/article/investigating-coupled-fluid-structure-acoustic-fsa-interaction/. [Online; accessed 07-2020]. 2020.

[89] Maziar Raissi et al. "Deep learning of vortex-induced vibrations." In: *Journal of Fluid Mechanics* 861 (2019), pp. 119–137.

[90] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations." In: *Journal of Computational physics* 378 (2019), pp. 686–707.

[91] Shengze Cai et al. "Physics-informed neural networks (PINNs) for fluid mechanics: A review." In: *Acta Mechanica Sinica* (2022), pp. 1–12.

[92] Guofei Pang, Lu Lu, and George Em Karniadakis. "fPINNs: Fractional physics-informed neural networks." In: *SIAM Journal on Scientific Computing* 41.4 (2019), A2603–A2626.

[93] Audrey Gaymann and Francesco Montomoli. "Deep neural network and Monte Carlo tree search applied to fluid-structure topology optimization." In: *Scientific reports* 9.1 (2019), pp. 1–16.

[94] Tal Ben-Nun and Torsten Hoefler. "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis." In: *ACM Computing Surveys (CSUR)* 52.4 (2019), pp. 1–43.

[95] Paolo Viviani et al. "Deep learning at scale." In: *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE. 2019, pp. 124–131.

[96] Bernhard Schölkopf. "Causality for machine learning." In: *Probabilistic and Causal Inference: The Works of Judea Pearl*. 2022, pp. 765–804.

[97] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. "Learning convolutional neural networks for graphs." In: *International conference on machine learning*. PMLR. 2016, pp. 2014–2023.

[98] Bernhard Gatzhammer. "Efficient and flexible partitioned simulation of fluid-structure interactions." PhD thesis. Technische Universität München, 2014.

[99] E.F Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer, 2009.

[100] Earl H Dowell and Kenneth C Hall. "Modeling of fluid-structure interaction." In: *Annual review of fluid mechanics* 33.1 (2001), pp. 445–490.

[101] F Liu et al. "Calculation of wing flutter by a coupled fluid-structure method." In: *Journal of aircraft* 38.2 (2001), pp. 334–342.

[102] D Lopes et al. "Influence of arterial mechanical properties on carotid blood flow: Comparison of CFD and FSI studies." In: *International Journal of Mechanical Sciences* 160 (2019), pp. 209–218.

[103] Rui Zhou et al. "Wind-induced nonlinear behaviors of twin-box girder bridges with various aerodynamic shapes." In: *Nonlinear Dynamics* 94.2 (2018), pp. 1095–1115.

[104] Zepeng Liu, Long Zhang, and Joaquin Carrasco. "Vibration analysis for large-scale wind turbine blade bearing fault detection with an empirical wavelet thresholding method." In: *Renewable Energy* 146 (2020), pp. 99–110.

[105] Chulin Yu, Zhiwen Ren, and Min Zeng. "Numerical investigation of shell-side performance for shell and tube heat exchangers with two different clamping type anti-vibration baffles." In: *Applied Thermal Engineering* 133 (2018), pp. 125–136.

[106] Fande Kong and Xiao-Chuan Cai. "A scalable nonlinear fluid–structure interaction solver based on a Schwarz preconditioner with isogeometric unstructured coarse spaces in 3D." In: *Journal of Computational Physics* 340 (2017), pp. 498–518. DOI: 10.1016/j.jcp.2017.03.043.

[107] Simone Deparis et al. "FaCSI: A block parallel preconditioner for fluid–structure interaction in hemodynamics." In: *Journal of Computational Physics* 327 (2016), pp. 700–718. DOI: 10.1016/j.jcp.2016.10.005.

[108] Fande Kong et al. "Simulation of unsteady blood flows in a patient-specific compliant pulmonary artery with a highly parallel monolithically coupled fluid-structure interaction algorithm." In: *International journal for numerical methods in biomedical engineering* 35.7 (2019), e3208. DOI: 10.1002/cnm.3208.

[109] Shunji Kataoka et al. "A parallel iterative partitioned coupling analysis system for large-scale acoustic fluid–structure interactions." In: *Computational Mechanics* 53.6 (2014), pp. 1299–1310. DOI: 10.1007/s00466-013-0973-1.

[110] Sam Hewitt et al. "OpenFPCI: A parallel fluid–structure interaction framework." In: *Computer Physics Communications* 244 (2019), pp. 469–482. DOI: https://doi.org/10.1016/j.cpc.2019.05.016.

[111] Fei Jiang et al. "A GPU-accelerated fluid–structure-interaction solver developed by coupling finite element and lattice Boltzmann methods." In: *Computer Physics Communications* 259 (2021), p. 107661.

[112] Alireza Naseri. "Developing numerical methods for fully-coupled nonlinear fluid-structure interaction problems." In: (2019).

[113] A. Naseri et al. "A semi-implicit coupling technique for fluid–structure interaction problems with strong added-mass effect." In: *Journal of Fluids and Structures* 80 (2018), pp. 94–112. DOI: 10.1016/j.jfluidstructs.2018.03.012.

[114] R.W.C.P. Verstappen and A.E.P. Veldman. "Symmetry-preserving discretization of turbulent flow." In: *Journal of Computational Physics* 187.1 (2003), pp. 343–368. DOI: 10.1016/S0021-9991(03)00126-8.

[115] F. X. Trias et al. "Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids." In: *Journal of Computational Physics* 258 (2014), pp. 246–267. ISSN: 00219991. DOI: 10.1016/j.jcp.2013.10.031.

[116] O. Estruch et al. "A parallel radial basis function interpolation method for unstructured dynamic meshes." In: *Computers and Fluids* 80 (2013), pp. 44–54. ISSN: 00457930. DOI: 10.1016/j.compfluid.2012.06.015.

[117] PD Thomas and CK Lombard. "Geometric conservation law and its application to flow computations on moving grids." In: *AIAA journal* 17.10 (1979), pp. 1030–1037. DOI: 10.2514/3.61273.

[118] A. J. Macleod. "Acceleration of vector sequences by multi-dimensional Δ^2 methods." In: *Communications in Applied Numerical Methods* 2.4 (1986), pp. 385–392.

[119] Alireza Naseri et al. "A second–order time accurate semi–implicit method for fluid–structure interaction problems." In: *Journal of Fluids and Structures* 86 (2019), pp. 135–155. DOI: 10.1016/j.jfluidstructs.2019.02.007.

[120] G. Karypis and V. Kumar. "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs." In: *SIAM Journal on Scientific Computing* 20.1 (1998), pp. 359–392. DOI: 10.1137/S1064827595287997.

[121] Richard Tran Mills et al. "Toward Performance-Portable PETSc for GPU-based Exascale Systems." In: *arXiv preprint arXiv:2011.00715* (2020).

[122] Victor Minden, Barry Smith, and Matthew G Knepley. "Preliminary implementation of PETSc using GPUs." In: *GPU solutions to multi-scale problems in science and engineering*. Springer, 2013, pp. 131–140.

[123] Paolo Crosetto et al. "Fluid–structure interaction simulation of aortic blood flow." In: *Computers & Fluids* 43.1 (2011), pp. 46–57.

[124] Yonghui Qiao et al. "Numerical simulation of two-phase non-Newtonian blood flow with fluid-structure interaction in aortic dissection." In: *Computer methods in biomechanics and biomedical engineering* 22.6 (2019), pp. 620–630.

[125] Romana Perinajová et al. "Assessment of turbulent blood flow and wall shear stress in aortic coarctation using image-based simulations." In: *Biomedical engineering online* 20.1 (2021), pp. 1–20.

[126] *2nd CFD Challenge Predicting Patient-Specific Hemodynamics at Rest and Stress through an Aortic Coarctation*. http://www.vascularmodel.org/miccai2013/. 2013.

[127] Miguel A Fernández, Mikel Landajuela, and Marina Vidrascu. "Fully decoupled time-marching schemes for incompressible fluid/thin-walled structure interaction." In: *Journal of Computational Physics* 297 (2015), pp. 156–181. DOI: 10.1016/j.jcp.2015.05.009.

[128] Nico Westerhof, Jan-Willem Lankhaar, and Berend E Westerhof. "The arterial windkessel." In: *Medical & biological engineering & computing* 47.2 (2009), pp. 131–141.

[129] Sanjay Pant et al. "A methodological paradigm for patient-specific multi-scale CFD simulations: from clinical measurements to parameter estimates for individual analysis." In: *International journal for numerical methods in biomedical engineering* 30.12 (2014), pp. 1614–1648.

Declaration of Authorship

I hereby declare that this thesis titled

Data-Integrated Methods for Performance Improvement of Massively Parallel Coupled Simulations

was independently completed.

Information taken directly or indirectly from external sources is properly marked as such.

Stuttgart, May 24, 2022