Data Transfer in Partitioned Multi-Physics Simulations: Interpolation & Communication

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

Florian Lindner

aus Kirchheimbolanden

Hauptberichter:	Prof. Dr. rer. nat. habil. Miriam Mehl
Mitberichter:	Prof. Dr. rer. nat. habil. Hans-Joachim Bungartz
	Prof. Dr. rer. nat. habil. Daniel Weiskopf

Tag der mündlichen Prüfung: 22. Juli 2019

Universität Stuttgart

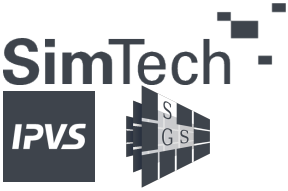
Institut für Parallele und Verteilte Systeme der Universität Stuttgart

2019

University of Stuttgart
Germany

Submitted to the University of Stuttgart

Involved institutions and departments:
Cluster of Excellence in Simulation Technology
Institute for Parallel and Distributed Systems
Chair of Simulation of Large Systems



Florian Lindner
University of Stuttgart
Universitätsstrasse 38
70569 Stuttgart
Germany

D 93 (dissertation)

Typeset using LaTeX and cover design by the author.

Although this thesis was written with utmost care, it cannot be ruled out that it contains errors. Please send any corrections and mistakes to `florian.lindner@xgm.de`.

Abstract

Partitioned multi-physics simulations allow to reuse existing solvers and to combine them to multi-physics scenarios. This provides not only greater flexibility and improved time-to-solution, but also helps to manage the increasing complexity of modern scientific software.

This thesis sees itself as a continuation of the works of B. Gatzhammer and B. Uekermann who developed a comprehensive tool to couple independent simulation codes. I focus on the two important aspects of interpolation between non-matching grids as well as communication between several parallel codes and conclude with aspects of software development of the coupling library preCICE.

The interpolation part puts special emphasis on radial-basis function interpolation. It starts with a thorough review of existing interpolation methods with special consideration of the black-box approach to multi-physics simulations and explores promising enhancements to RBF interpolation. Numerical experiments provide a rigorous testing for accuracy, stability and scaling behavior of different variants of RBF implementations. Following the insights gained from the numerical experiments, a highly-optimized parallel implementation for preCICE is developed, containing various measures to improve accuracy and stability of the interpolation.

The communication part first defines the requirements for partitioned simulations in terms of communication. A new technique for peer-to-peer communication networks between distinct MPI domains is developed and evaluated against existing approaches. Furthermore, a fast method to establish connections via the file system is presented. Both measures optimize the initialization phase and achieve a considerable speedup. Finally, a strategy to fully decouple algorithmically independent participants on the communication protocol level is implemented and tested.

In the last part, the software-related challenges in developing a parallel scientific application involving multiple independent solvers are outlined. I show how the preCICE project handles testing, profiling and integration of a large parallel scientific software with multiple participants. A profiling library for distributed applications has been developed and is extensively used in preCICE and potentially other projects.

Kurzzusammenfassung

Partitionierte Mehr-Physik Simulationen erlauben es vorhandene Simulationsprogramme wiederzuverwenden und zu komplexen Mehr-Physik Szenarien zu kombinieren. Damit wird nicht nur eine größere Flexibilität und kürzere Entwicklungszyklen erreicht, ebenfalls erlaubt es bessere Beherrschung der hohen Komplexität moderner wissenschaftlicher Software.

Diese Arbeit kann als Fortführung der Arbeiten von B. Gatzhammer und B. Uekermann verstanden werden. Diese entwickelten eine umfassende Software zum Koppeln mehrerer unabhängiger Simulationsprogramme. In dieser Arbeit konzentriere ich mich auf zwei wichtige Aspekte, der Interpolation zwischen nicht-übereinstimmenden Gittern und der Kommunikation zwischen mehreren parallelen Programmen. Abschließend gebe ich einen Überblick über Software-Entwicklung der Kopplungsbibliothek preCICE.

Die Interpolation konzentriert sich auf die Methode der radialen Basisfunktionen. Ausgehend von einem umfassenden Überblick bestehender Methoden mit spezieller Berücksichtigung des Black-Box Ansatzes bei Mehr-Physik Simulationen, werden vielversprechende Erweiterungen der RBF Interpolation behandelt. Mithilfe von numerischen Experimenten werden diese auf Genauigkeit, Stabilität und Skalierungsfähigkeit untersucht. Die gewonnen Erkenntnisse wurden anschließend in einer hoch-optimierten, parallelen Implementierung der Interpolation für preCICE umgesetzt.

Zur Kommunikation werden zuvorderst die Anforderungen partitionierter Simulationen definiert. Eine neue Herangehensweise um peer-to-peer Netzwerke zwischen getrennten MPI-Domänen zu realisieren wird entwickelt und getestet. Weiterhin wird eine neue Methode um den Verbindungsaufbau über das Dateisystem zu beschleunigen gezeigt. Beide Methoden tragen dazu bei, die Initialisierungsphase der Simulation erheblich zu beschleunigen. Ebenso wird eine Methode gezeigt und umgesetzt die algorithmisch unabhängige Kopplungsteilnehmer auch auf der Kommunikationsebene entkoppelt.

Im letzten Teil dieser Arbeit werden die Herausforderungen beim Entwickeln einer wissenschaftlichen Anwendung die mehrere Löser umfasst dargestellt. Ich zeige, wie das Testen von Software, Profiling und die Integration von preCICE mit mehreren Simulationsprogrammen gelöst wurde. Dabei wird eine Bibliothek zum Profilen von verteilten Anwendungen entwickelt und eingesetzt.

Acknowledgments

In science, they say, you are standing on the shoulder of giants[1]. While one usually thinks about the uber-giants from former times, such as Turing, Newton or Darwin, I also want to thank the millions of little giants who make me see further, even if their names are long forgotten.

In particular, it has been a great honor to not only stand on their shoulders, but at the same time standing side by side with some of these giants and having the opportunity to meet them at conferences, to work with them in projects and to receive advice and guidance from them.

I would like to thank everyone, who contributed directly or indirectly to this thesis, and some of them in particular.

My advisor Prof. Miriam Mehl for discussions, guidance and putting things back into perspective and especially for an always open door. Prof. Hans Bungartz not only being my co-examiner but who also forwarded my application as a doctoral candidate from Munich to Stuttgart and is responsible for the great time I have.

My colleagues at the University of Stuttgart, making our workplace a fun and stimulating place for computer science and far beyond.

The entire preCICE project team for being part of a small academic project which grew into a viable open-source project.

All this would never be possible, without those giants that produce the stack of open-source software on which almost any scientific software project is based.

Not only in science you are standing on the shoulders of giants, likewise in your entire life you can never stand as a solitary.

A big thank you goes out to my family, my parents, my girlfriend and her parents for creating moments of vacation to guide me out of the rabbit hole.

Last but not least, I give thanks to the gift of music, that focused my thoughts when they went astray.

[1] These giants obviously also introduced the tradition of writing an acknowledgments section, to which I will obey here.

Contents

List of Symbols and Acronyms

Symbols

Symbol	Meaning
Γ	Interface Mesh
Ξ	Input Mesh
ξ	Input Data Site
Δ	Output Mesh
δ	Output Data Site
Ω	Problem Domain
φ	Basis Function
λ	Basis Function Weighting Factor
s	RBF Interpolant Operator
h	Mesh Width
H	Mapping Operator
d	Dimensionality
u	Velocity
p	Pressure
F	Force
ρ	Density
ν	Kinematic Viscosity
μ	Shear Modulus
ν_s	Poisson's Ratio
Re	Reynolds Number
St	Strouhal Number
t	Time as measured by the system clock
$\tilde{t}$	Time as measured by the steady clock
$\mathbb{R}^d$	Real Numbers in d-dimensional space
$\mathbb{P}^n$	Polynomial space of degree n
L_∞	Infinity Norm
$\mathcal{O}(f(x))$	Big-$\mathcal{O}$ Landau notation

Acronyms

Acronym	Meaning
CI	Continuous Integration
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
CSM	Computational Structure Mechanics
FSI	Fluid-Structure Interaction
FLOPS	Floating Point Operations per Second
GPFS	IBM General Parallel File System
HPC	High-Performance Computing
HLRS	Höchstleistungsrechenzentrum Stuttgart
IP	Internet Protocol
I/O	Input/Output
LB	Lattice-Boltzmann
LES	Large Eddy Simulation
LRZ	Leibniz-Rechenzentrum
M2N	Many-to-Many Communication
MPI	Message Passing Interface
MQ	Multi-Quadrics Basis Function
RBF	Radial-Basis Function Interpolation
RMSE	Root Mean Squared Error
STL	Stereolithiography Format
TCP	Transmission Control Protocol
TPS	Thin-Plate Splines Basis Function
UDP	User Datagram Protocol
URANS	Unsteady Reynolds Averaged Navier Stokes
UUID	Universally Unique Identifier
VS	Volume Splines Basis Function
VTK	Visual Toolkit

1 Partitioned Multi-Physics Simulations

> If anything, there's a reverse Moore's Law observable in software: As processors become faster and memory becomes cheaper, software becomes correspondingly slower and more bloated, using up all available resources.
>
> Jaron Lanier

1.1 Introduction

In spite of the dystopian quote I decided to start this chapter with, science and society expect results from the ever growing computing power available to us. Results that give insight into the mechanics of climate change or aortic dissection, into the dynamics of a flapping wing or an artificial fish.

There is nothing easier in simulation sciences than burning ever more computing power. By turning up spatial or temporal resolution, the complexity of a given simulation case can easily be tuned to use up all available computing power. However, your mileage may vary. For simulations that are already converged to a validated result, the insight gained from increasing the resolution is minimal. As Uekermann put it in his thesis [105], more computing power does not *automatically* lead to more resolved physical effects. There is much work to be done turning the surplus of computing power into adequate results or, to put it in another way, keeping up the science per FLOP.

Multi-physics simulations are a way to actually turn the investment of new and faster computers into novel scientific and engineering insight. In reality, more often than not, more than one "physics" actually drive a concrete problem. From a physicist or philosophers point of view, there is just one physics. From a methodological and simulation perspective, there are many. High-speed fluid dynamics warrant a different simulation approach than

static structural mechanics. This change of perspective yields two different approaches that multi-physics simulations can be distinguished into.

The *monolithic* approach, on one hand, uses a single system of equations to describe and solve the coupled problem. On the other hand, *partitioned* methods use existing single-physics solvers and couple them to a simulation that strives to solve the overall coupled problem.

The monolithic approach can be more efficient and robust for a single simulation case. After all, the solver "knows" about the physics involved and this knowledge can consequently be integrated into the solver. However, the monolithic architecture makes the integration of new physics or the adaption to vastly different simulation cases hard. A partitioned approach fosters modularization and with that easier exchange of principal components of the simulation. As such, the partitioned approach can be one way to break the vicious cycle suggested by Jaron Lanier in the opening quote. The modularization of the partitioned approach can help to manage the increasing complexity of simulations involving more and physical effects.

Three main areas of interest can be identified in a partitioned multi-physics simulation:

- **Equation coupling** is responsible for steering the coupled simulation, i.e., determining convergence and minimal time step sizes as well as controlling the subcyling of coupling iterations in one time step. Furthermore, acceleration schemes such as quasi-Newton can be used to speed up convergence of the iterative solution of the coupling equations.

- **Data interpolation** or mapping is crucial for transferring data between solvers that use different meshes at the coupling interface. This is usually the case, as each mesh is adapted to the solver's respective problem domain.

- **Efficient communication** takes place between solvers as well as between processors of the same solver. Avoiding expensive all-to-all communication patterns requires setting up a sparse communication network between solvers. In partitioned simulations, communication is a special challenge, as each solver can bring its own communication model and framework, which might be challenging to integrate with.

In this thesis, I focus on the two latter fields, interpolation and communication. Equation coupling and quasi-Newton schemes as used in the coupling library preCICE are covered in [9, 70] or in broader context in the doctoral thesis of Klaudius Scheufele [92].

1.2 The ExaFSA Project

The ExaFSA "Exascale Simulation of Fluid-Structure-Acoustics Interactions" project is part of the DFG priority program 1648 "Software for Exascale Computing" (SPPEXA)[1].

The SPPEXA project works as an umbrella organization for a number of more or less independent projects. The overarching mission is to develop algorithms, system and application software as well as tools for programming and data exploration for the field of future high-performance computing. Upcoming systems with more than 10^7 processing units and a compute power ranging up to exaflops, i.e., more than 10^{18} floating point operations per second require to transition from sequential or moderately parallel to massively parallel processing. The various sub-groups of the SPPEXA project provide software from infrastructure to application level to leverage this transition.

The objective of the ExaFSA project is to develop efficient massively parallel simulations of fluid-structure-acoustics interactions. It takes a modular approach and, thus, the developed methods and software are going to be re-usable for other coupled multi-physics simulations.

Project partners within the ExaFSA project are the University of Siegen contributing with the fluid and acoustic solver Ateles [113], the Technical University of Darmstadt with the fluid and acoustic solver FASTEST [68, 72], Tohoku University in Japan with the code translation framework Xevolver [99] and the University of Stuttgart together with the Technical University of Munich with the coupling library preCICE [21].

1.3 The preCICE Project & Software

preCICE (Precise Code Interaction Coupling Environment) is a library that facilitate connecting multiple single-physics simulation codes. This way, complex multi-physics scenarios can be constructed in a plug-and-play manner.

The project was started by Bernhard Gatzhammer as part of his PhD thesis [50] and further pursued by Benjamin Uekermann [105]. The current development team consists of G. Chourdakis, B. Rüth, F. Simonis, A. Totounferoush, B. Uekermann and myself.

The software is open-source, licensed under the GNU Lesser General Public License (LGPL) version 3 [51]. The development takes place in an open and public process, much of the project coordination happening at GitHub[2]. As of April 2019, the main repository contains almost 2700 commits from 23 authors.

[1] `http://www.sppexa.de`
[2] `https://github.com/precice/`

1.3.1 Structure and Design

preCICE is designed as an object-oriented application, written in C++. It tries to leverage the features provided by the C++11 standard to the fullest and as such requires a moderately modern compiler.

preCICE is not used as a standalone application but as a library. It is used by means of solver-specific adapter codes, that translate data structures such as meshes and vertex data and simulation steering commands between the solver and the preCICE library, see also Fig. 1.3. How such an adapter is implemented and integrated with the solver depends on the solver and very much on the kind of API provided by the solver as well as the openness of the code. Open-source solvers usually integrate the calls to preCICE directly in their main solver loop.

The software is organized in packages, reflected in a sub directory for each package. The following packages exist inside preCICE (in alphabetical order).

action defines callback functionality that can be used to call pre-defined functions or arbitrary Python code at certain points of the simulation. These functions can, e.g., modify data before they are sent to the coupling partner.

com provides abstractions for MPI and socket communication to general communication routines. For more information see Sec. 5.2.1.

drivers holds the main function for building the preCICE executable.

io implements exporting functionality of the coupling meshes and data to, e.g., VTK files.

logging contains logging routines used by all packages. The logging is implemented based on the Boost Logging library.

m2n defines the parallel communication between the participants based on the abstract routines from the *com* package.

math provides general mathematical routines and constants. This also includes comparison functions for floating point numbers.

mapping implements the nearest-neighbor, nearest-projection and radial basis-function interpolation for mapping exchanged data between the participant's grids. This functionality is covered in depth in Chapters 2 to 4.

mesh holds methods and data structures for mesh and coupling data. It is used by almost every other package.

partition computes the partitioning of meshes at the initialization of the simulation, based on the local partitioning in the involved parallel solvers.

query holds functionality to query for certain points inside a mesh.

testing provides the framework for writing serial or parallel unit as well as integration tests, see Sec. 6.2.

utils collects general functionality that is used by most packages and does not warrant its own package.

xml is the XML configuration framework and abstraction to the *libxml2* XML parsing library.

Each package follows a uniform directory layout and contains the following sub directories:

config/ contains callback functions that configure the package. These callbacks are set up for certain XML tags and called when these tags are encountered in the configuration file. It enables a decentralized and self-contained configuration for each package.

impl/ contains functionality that is not part of the public interface of the package.

tests/ contains unit tests that automate testing the functionality contained in the package. As unit tests, these tests focus on testing the functionality directly provided by the package in an isolated environment. They are supplemented by integration tests, provided by the *precice* package.

Figure 1.1 shows the dependency relation of the packages which provide the preCICE core functionality.

1.3.2 The Distributed Architecture of preCICE

In the first evolutionary step, as developed and implemented by Gatzhammer in [50], preCICE was implemented as a centralized architecture. One server instance as a central aggregation point was launched alongside with the processes of each solver, as shown in Fig. 1.2. In the initialization phase of a coupled simulation, the server collects the interface meshes from each solver process and reconstructs the full interface mesh. During the simulation, all coupling related computation is performed on the server and communication is relayed through it. For simulation runs that comprise tens of thousands of processes, it

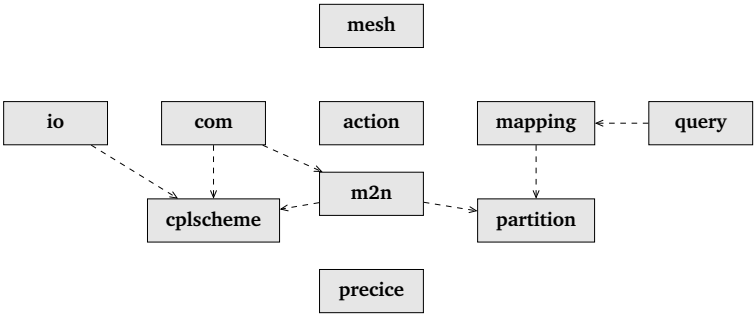


Figure 1.1: Dependency of the principal preCICE packages. Every package depends on the mesh package. On the other hand, the *precice* package depends on every other package. The dependency edges for these two packages are therefore omitted for brevity. Dependencies that exist only inside the *tests* sub-package are ignored.

quickly becomes obvious, that this application architecture is inherently not scalable to a sufficient degree.

Before I continue to dive into the details of the current distributed architecture, a word on nomenclature. We prefer the term *distributed* vs. *centralized* in contrast to *parallel* vs. *sequential*. While the terms are often confused and a clear distinction is not easy, [83] aims for a concept formation. He states, that parallelism's fundamental issue consists in mastering *efficiency*, whereas distributed computing is about mastering *uncertainty*. In distributed computing, each entity knows only about a subset of the whole set of inputs. Furthermore, the geographical[3] distribution of the problem is not a design choice, but is an input parameter.

These key concepts outlined above align very well with the challenges faced during the transition of preCICE to a distributed application architecture and the resulting design decisions made in the second evolutionary step. For instance, each rank holds only a portion of the interface mesh. These challenges, posed by the distributed way information is held amongst the ranks, will be detailed in the following.

In [105], preCICE was re-designed to become a distributed application, as far as possible devoid of a central command, control, communication or computation instance. The main goal, as stated by Uekermann, is not a classical speed-up, but rather rendering a central instance unnecessary, or, in other words, to not get into the way of the scalability of the

[3]here, geographical refers to the distribution of the problem set among different ranks and nodes

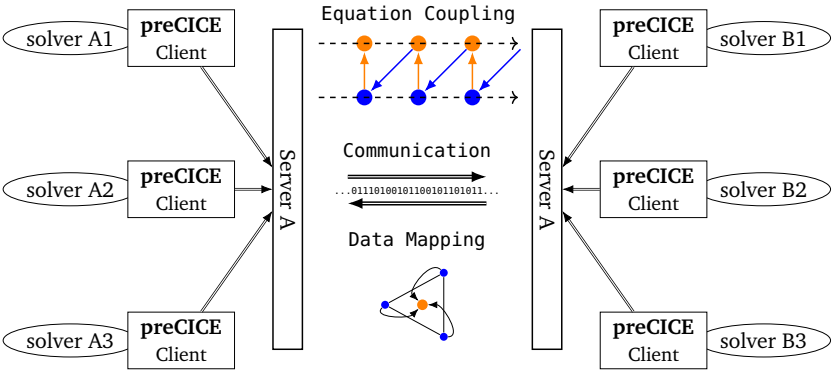


Figure 1.2: Old communication scheme of preCICE as implemented in [50]. For each participant, one server instance of preCICE was launched to perform coupling related computation and communication.

involved solvers.

Figure 1.3 shows the new communication scheme of preCICE. The most obvious difference is the absence of a server component. However, there is still a *master* for each solver who acts as a primus inter pares. One rank of each participant runs in master mode, whereas all the other ranks run in slave mode. The master is currently hard-coded to rank 0, which is however an arbitrary decision and can be adapted easily. From a user perspective, this choice is transparent, i.e., all ranks appear identical and each can potentially hold a part of the mesh.

Given the two participants A and B with the number of ranks p_A and p_B, the two coupling meshes are defined as

$$\bigcup_{i=1}^{p_A} \Gamma_A^i = \Gamma_A, \quad \bigcup_{i=1}^{p_B} \Gamma_B^i = \Gamma_B \tag{1.1}$$

where Γ_A^i, Γ_B^i denote the local parts and Γ_A, Γ_B the complete solver meshes as the union of all local parts. As mentioned above, the decomposition of the meshes is supplied by the participants A and B and can not be influenced by preCICE. The local meshes are not necessarily disjoint, a vertex with the same coordinates might be defined multiple times, e.g, due to ghost layers in the participant's partitions. This case is handled by treating the two vertices as separate, albeit with same coordinates. While this usually is not problematic, it might introduce undefined behavior for some cases, such as nearest-neighbor mapping selecting a value from a random one of the identical vertices.

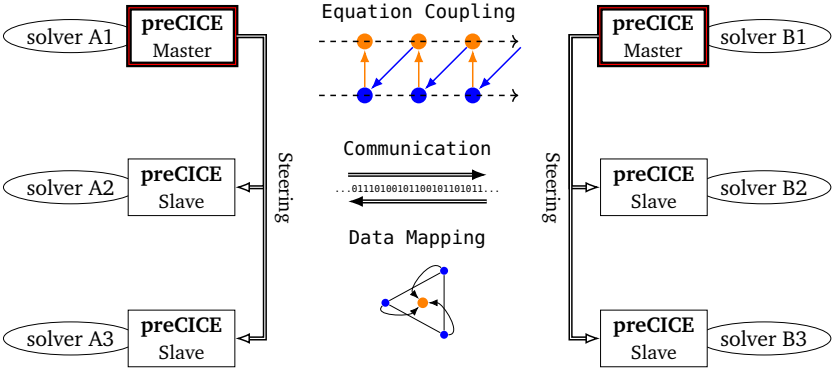


Figure 1.3: New communication scheme without a central server, as designed in [105]. A *master* rank acts as a steering instance for tasks which are not yet parallelized. Outside the initialization phase, all communication happens in a peer-to-peer fashion directly between slaves.

The mapping or interpolation of values from one mesh to another happens at one of the participants, as defined in the configuration file. In order to perform that mapping, the respective participant has to have knowledge of both of the meshes Γ_A and Γ_B. Without loss of generality we assume that the mapping happens at participant A and, thus, Γ_B needs to be transferred from B to A. At A, each slave computes the vertices from B it requires to perform the mapping. Based on this information, Γ_B is *re-partitioned* at A. Two different strategies, called *broadcast/filter* and *pre-filter/post-filter* are available and can be selected in the configuration file. Figure 1.4 visualizes this process in the *broadcast/filter* variant, Tab. 1.1 contrasts the two variants.

The selected mapping plays an important role in the re-partitioning process, as it determines in step IV which vertices are required for a processor-local portion of the mapping to take place. The technical details of this are described in Sec. 3.4.2.

Both variants posses their individual advantages and disadvantages, which are discussed in [105, p. 73]. Whereas the *broadcast/filter* variant features a computationally expensive filtering of the entire mesh (step IV), the *pre-filter/post-filter* variant first does a pre-filtering using the bounding box for each slave's partition, which reduces the computational load. On the other hand, step III in the *pre-filter/post-filter* variant is sequential, which is a potential bottleneck in simulations with a large number of processors. Due to the broadcasting in step III of the *broadcast/filter*, it benefits from efficient broadcast routines as the ones

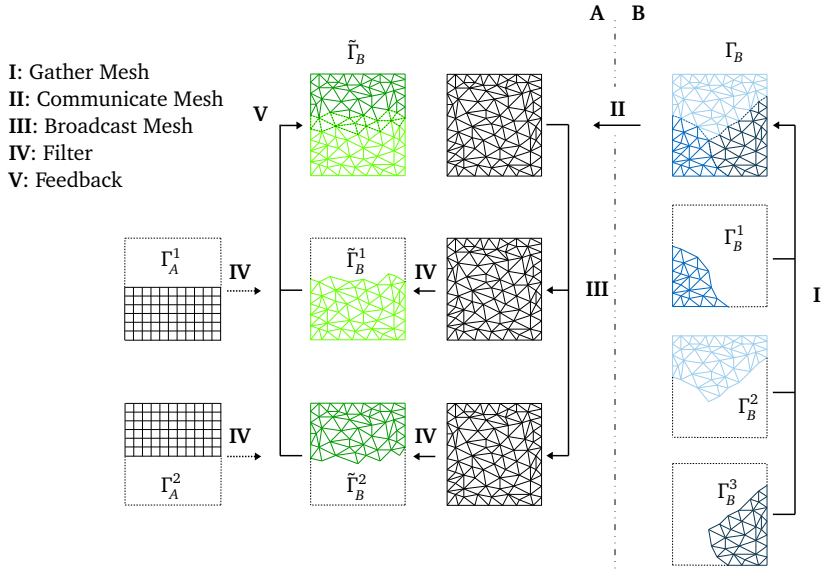


Figure 1.4: Re-partitioning of meshes, in the *broadcast/filter* variant. Participant A (left) and B (right) run on three and four processors, respectively. Only local mesh contributions Γ^i from each processor i are given to preCICE. To allow for a mapping to take place at participant A, Γ_B needs to be available at A. In step I, $\Gamma_B = \bigcup_i \tilde{\Gamma}_B^i$ is gathered at the master process of B and then sent to the master process of A (II). At A, Γ_B is now broadcasted to all slaves (III) and filtered locally by each slave (IV). This results in a re-partitioning $\tilde{\Gamma}_B$ of Γ_B. The information containing the new partitioning is finally send back to the master of A (V). Note that, for sake of simplicity, both masters do not hold a mesh here and slaves can as well hold no mesh information. Figure courtesy of Benjamin Uekermann [105]

.

	broadcast/filter		pre-filter/post-filter
I	M_B gathers Γ_B	**I**	M_B gathers Γ_B
II	M_B transfers Γ_A to B	**II**	M_B transfers Γ_A to B
III	M_A broadcasts Γ_B to each S_A^i	**III**	Every S_A^i sends a bounding box round Γ_A^i to M_A. M_A filters Γ_B accordingly and distributes the filtered mesh to all S_A^i (*pre-filter*).
IV	Each S_A^i filters Γ_B based on the mapping method	**IV**	Each S_A^i filters Γ_B based on the mapping method (*post-filter*).
V	M_A gathers the mesh distribution information from all S_A^i	**V**	M_A gathers the mesh distribution information from all S_A^i

Table 1.1: Mesh re-partitioning strategies

usually found in MPI implementations.

The technical realization of the communication, i.e., the underlying protocols such as MPI or TCP/IP are detailed further in Chapter 5.

1.3.3 Applications

In this section, I quickly summarize four publications as examples how preCICE is used by researchers outside the SPPEXA ExaFSA group. They include classical fluid-structure interaction problems and a fluid-fluid coupling between diverse discretization schemes and turbulence models.

In [96], preCICE is used in the field of nuclear reactor simulations. Turbulence induced vibrations from the cooling fluid surrounding the fuel rods are the main source of excitation of the rods. Sharma, Santis, and Shams describe an extension to the standard URANS (Unsteady Reynolds Averaged Navier Stokes) turbulence model that more accurately models these vibrations. The strongly-coupled fluid-structure system is simulated by coupling a finite-volume fluid with a finite-element structure solver. The solvers are coupled by preCICE and the resulting framework is described as “very flexible and powerful” by the authors. The fluid-structure simulations use preCICE’s implicit IQN-ILS (Interface Quasi-Newton with Inverse Jacobian from Least-Squares) to accelerate the convergence between the two black-box solvers, resulting in much faster convergence than classical Gauss-Seidel methods. The mapping of values between the coupling interfaces is achieved by using nearest-neighbor interpolation. The resulting setup is validated against a number of test cases, including the well-known Turek [103] test case, showing good agreement with the

experimental results.

Santasmasas et al. use preCICE to implement a hybrid RANS-LES method in [89]. In contrast to the classical case of fluid-structure interaction, mentioned in the previous paragraph, the coupling is performed between two fluid regions, which use URANS and LES as their respective turbulence modeling methods. The solvers coupled are a Lattice-Boltzmann (LB) solver using LES turbulence modelling, running on a GPU and an OpenFOAM based Navier-Stokes solver with a URANS turbulence model, running on CPUs. As both solvers posses a region of overlap, preCICE was modified to allow for transferring both data from boundary cells as well as internal cells in the overlap region. The solvers are executed simultaneously using the parallel-explicit coupling method from preCICE. Their research is still at the early stages, subsequent research will also include evaluating further coupling schemes, such as implicit ones.

Another classical fluid-structure interaction application is demonstrated in [26]. Cinquegrana and Vitagliano use preCICE to couple a URANS flow solver called ZEN with the open-source structural solver CALCULIX[4]. The decision for preCICE as a coupling library is driven by the open-source environment of the project, its ease of implementation and learning curve for using the structural solver and the availability of both explicit and implicit coupling methods as well as the choice of mapping algorithms. To interface CALCULIX with preCICE, the existing adapter[5], developed in the preCICE project is used. For the ZEN fluid solver, a new adapter to preCICE is developed. The coupled setup is first verified using an oscillating flap and subsequently applied to a supersonic panel flutter problem. The cases use RBF with thin-plate splines and nearest-neighbor interpolation, respectively. The publication concludes with encouraging, though only preliminary results.

A more unusual scenario is investigated by Luo et al. In [71], a bio-inspired fish is modeled using an in-house Navier-Stokes solver coupled via preCICE to the finite-element solver CALCULIX. The effect of fin flexibility on the propulsive performance of the fish-model is evaluated. For that, a numerical validation using two FSI examples is performed. First, a scenario similar to the cylinder flap, presented in Sec. 4.10 is validated in terms of shedding frequency. Second, the bending of a three-dimensional flexible plate in uniform flow is evaluated. For the actual simulation, conservative radial-basis function interpolation is used for data transfer and IQN-ILS quasi-Newton methods to accelerate the coupling iterations.

[4] http://www.calculix.de
[5] https://github.com/precice/calculix-adapter

1.4 Contributions

As mentioned above, this thesis focus on the aspects of interpolation and communication for partitioned multi-physics simulations. In a nutshell, the main contributions of this thesis are

- Provide an exhaustive overview of the theoretical aspects of interpolation with radial-basis functions with special consideration of the black-box coupling approach. This includes the evaluation of various solution approaches for fundamental problems of this type of interpolation.
- Two different methods for algorithmically treating the RBF interpolation augmented by a polynomial for either consistent or conservative implementation are developed.
- In order to provide a meaningful assessment of the error for conservative interpolation, two new error metrics are presented.
- Numerous variants of the RBF interpolation algorithm and parameters are scrutinized for accuracy and stability for a large number of parameters.
- A highly-optimized parallel implementation of the RBF interpolation for preCICE is developed. This also involves the efficient and reusable integration of a tree-based search structure of spatial queries in preCICE. The efficiency of the implementation is validated using weak and strong scaling.
- Implementations of MPI on the HazelHen and SuperMUC HPC systems are evaluated for suitability to create peer-to-peer communication methods. A new method to establish a communication between coupling participants using MPI is implemented and evaluated.
- A framework to effortless instrument and profile partitioned, multi-node and multi-participant simulations is developed.
- A testing framework ranging from atomic unit tests with special consideration of parallel applications to multi-participant system tests is implemented. Furthermore, a flexible tool chain for numerical experiments involving preCICE is created.

1.5 Structure of this Thesis

Chapter 2: **Interpolation Methods for Multi-Physics Simulations** gives an introduction to interpolation methods found in multi-physics simulation software. Interpolation using radial-basis functions is presented in detail. The chapter gives an overview on current research regarding convergence and stability properties as well as different methods to improve weak spots of RBF interpolation.

The implementation of the RBF algorithm is presented in Chapter 3: **Optimization & Implementation of RBFs in preCICE**. The chapter details the actual implementation of the algorithm using the PETSc library. It also discusses how a spatial index to facilitate fast construction of the linear systems is integrated in preCICE.

In the final chapter about interpolation, Chapter 4: **Numerical Experiments & Results**, numerous variations of the algorithm and the parameters are evaluated for stability and accuracy. The C++ implementation of the interpolation algorithm is tested for scalability. Furthermore, different linear solvers and preconditioners contained in PETSc are tested for their suitability to solve the resulting linear systems.

Chapter 5: **Efficient Communication on HPC Machines** introduces how preCICE uses different communication channels and the resulting requirements. The implementation status of TCP/IP and MPI at the HazelHen and SuperMUC HPC systems are charted. A new method to exchange the initial connection tokens is presented and benchmarked. Finally, the improved implementation of MPI communication in preCICE is evaluated on the two HPC systems.

In Chapter 6: **Software Development** is about discussing two important aspects in the development of preCICE. First, the challenges of profiling a distributed, partitioned simulation application are outlined. For that, a new, reusable framework that eases profiling and instrumentation as well as visualization of the results of such types of applications is developed. Second, the important aspect of automated software testing also holds special challenges for distributed and parallel applications. It is explained how the preCICE project tackles automated testing from unit tests to system tests of cases involving multiple participants.

2 Interpolation Methods for Multi-Physics Simulations

> The purpose of computing is insight, not numbers.
>
> Richard Hamming

In partitioned multi-physics simulations each solver not only uses algorithms that are tailored for the respective physical domain, but also utilizes appropriate meshes. Fluid simulations (CFD) typically require much finer mesh resolutions than structural simulations (CSM) and special features such as highly resolved boundary layers.

Therefore, one of the crucial parts of a partitioned multi-physics coupling is data interpolation between non-matching meshes. Due to our black-box approach, we assume that the mesh is represented as a point cloud, i.e., without surface triangulation. While solvers can optionally provide such a triangulation by defining basic primitives such as edges, triangles and quads, the coupling library cannot rely on it. Furthermore, the missing information about the solver's discretization scheme, mesh topology and shape functions restricts the choice of applicable data mapping algorithms.

Several interpolation methods that work on point cloud data exist, of which interpolation by radial basis functions is among the most commonly used. In this chapter, I will first compare RBF interpolation methodologically briefly with alternative interpolation methods. The interpolation quality and numerical condition on different meshes as well as test functions will be evaluated in Chapter 4.

Special emphasis will be given to the well-known fact, that radial basis function interpolation produces ill-conditioned systems, which harms the solution accuracy and the time-to-solution, in particular for large systems. Methods to cope with this are reviewed in this work. These methods include basis transformations, preconditioning, re-scaling of the interpolant and an alternative numerical treatment of the additional polynomial that is used to properly capture globally constant or linear components of the data.

2.1 Problem Formulation

The general, high-level problem setting consists of two independent solvers with two problem domains. A coupling interface Γ connects the two domains. While the two domains are mostly disjoint, a small overlap occurs in practice and should not pose a problem for the interpolation. Over the interface arbitrary quantities are exchanged. In a fluid-structure interaction setting, as an example, these quantities are usually displacements from the structure to the fluid solver and pressures or forces vice-versa. The quantities to be exchanged usually live on different meshes, therefore a mapping or interpolation between the source and the target mesh is needed.

Without loss of generality, this problem formulation for mapping focuses on the specific case of fluid-structure interaction, because some concepts such as *virtual work* only make sense this way. The results are however, independent of the concrete application. In this section, I denote the fluid side with the index f and the structure side with the index s. This introduction follows [108].

We consider the continuous vector field of displacement values $\boldsymbol{u}(\boldsymbol{x})$ and the scalar field of pressure values $p(\boldsymbol{x})$, with $x \in \Gamma$. The kinematic and dynamic coupling conditions at the interface Γ are

$$\begin{aligned} \boldsymbol{u}_f(\boldsymbol{x}) &= \boldsymbol{u}_s(\boldsymbol{x}), \\ p_s(\boldsymbol{x}) &= p_f(\boldsymbol{x}) \end{aligned} \tag{2.1}$$

In the following, I consider only scalar fields, since vector fields are mapped component wise in the same way for each dimension.

The continuous fields are approximated by still continuous fields u^h and p^h, respectively, in a suitable basis representation with coefficient vectors $\bar{u}$ or $\bar{p}$:

$$u_f(\boldsymbol{x}) \approx u_f^h(\boldsymbol{x}) = N_f^T(\boldsymbol{x})\,\bar{u}_f \tag{2.2}$$

$$p_f(\boldsymbol{x}) \approx p_f^h(\boldsymbol{x}) = N_f^T(\boldsymbol{x})\,\bar{p}_f \tag{2.3}$$

$$u_s(\boldsymbol{x}) \approx u_s^h(\boldsymbol{x}) = N_s^T(\boldsymbol{x})\,\bar{u}_f \tag{2.4}$$

$$p_s(\boldsymbol{x}) \approx p_s^h(\boldsymbol{x}) = N_s^T(\boldsymbol{x})\,\bar{p}_s \tag{2.5}$$

where N_f and N_s are vectors of shape functions, $\bar{u}$ the nodal displacements and $\bar{p}$ the nodal pressure values. In a black-box coupling scenario, only $\bar{u}$ and $\bar{p}$ are known to the coupling component. The bases $N_f(\boldsymbol{x})$ and $N_s(\boldsymbol{x})$, on the other hand, remain opaque, as they depend on the discretization method used for displacements or pressure.

The nodal forces on the fluid- and the structure mesh can be obtained from the pressure values by integration

$$\begin{aligned} \bar{F}_f &= \int_{\Gamma_f} N_f(\boldsymbol{x}) p_f^h(\boldsymbol{x}) \, \mathrm{d}\Gamma = \int_{\Gamma_f} N_f(\boldsymbol{x}) N_f^T(\boldsymbol{x}) \bar{p}_f \, \mathrm{d}\Gamma = M_{ff} \, \bar{p}_f \\ \bar{F}_s &= \int_{\Gamma_s} N_s(\boldsymbol{x}) p_s^h(\boldsymbol{x}) \, \mathrm{d}\Gamma = \int_{\Gamma_s} N_s(\boldsymbol{x}) N_s^T(\boldsymbol{x}) \bar{p}_f \, \mathrm{d}\Gamma = M_{ss} \, \bar{p}_s. \end{aligned} \tag{2.6}$$

The mapping H is a linear operator that translates from values located at nodes of one mesh to nodes at the other mesh. We assume - without loss of generality - that we map the displacements $\bar{u}_s$ from structure (index s) to fluid (index f) via an operator H_{fs} and H_{sf} vice versa.

$$\bar{u}_f = H_{fs} \, \bar{u}_s \tag{2.7}$$

$$\bar{F}_s = H_{sf} \, \bar{F}_f \tag{2.8}$$

This defines a direct or *consistent* mapping if the row-sum of H_{fs} is equal to 1, i.e., a constant field of values is mapped exactly from one mesh to another. Consistent mapping is usually used for displacements and pressures.

Conservation of energy at the interface is expressed as

$$\int_{\Gamma} u_f(\boldsymbol{x}) \, p_f(\boldsymbol{x}) \, \mathrm{d}\Gamma = \int_{\Gamma} u_s(\boldsymbol{x}) \, p_s(\boldsymbol{x}) \, \mathrm{d}\Gamma. \tag{2.9}$$

The fluid- and the structure solver can use different discretizations of Γ, the fluid interface mesh Γ_f and the structure interface mesh Γ_s, respectively.

The discrete form of the energy conservation Eq. (2.9) at the interface can be derived as,

$$\bar{u}_f^T \, \bar{F}_f = \bar{u}_s^T \, \bar{F}_s. \tag{2.10}$$

The *conservative* form of interpolation can be directly derived from the discrete formulation of energy conservation Eq. (2.10)

$$\bar{F}_s = H_{fs}^T \, \bar{F}_f. \tag{2.11}$$

For pressure values, this gives the conservative mapping

$$\bar{p}_s = M_{ss}^{-1} \, H_{fs}^T \, M_{ff} \, \bar{p}_f, \tag{2.12}$$

if H_{fs} is a consistent mapping. This obviously requires knowledge of M_{ss} and M_{ff}. In order to ensure conservation of virtual work in a black-box coupling environment, the mapping has to be restricted to $\bar{u}$ and $\bar{F}$.

The conservative mapping is usually used for integral values, such as forces and ensures that

$$\sum_{i}^{\Gamma_f} \bar{F}_{f,i} = \sum_{i}^{\Gamma_s} \bar{F}_{s,i}. \tag{2.13}$$

From the transposition of H_{fs} it is obvious, that the column-sum of a conservative mapping matrix H_{fs}^T is equal to 1 and, furthermore, that a consistent interpolation can be made conservative by transposition and inverting the mapping direction and vice-versa.

Note that though for each of the following interpolation methods such a mapping matrix H_{fs} can be defined, it is usually never created explicitly.

While obviously a mapping H_{fs} or H_{sf} can be both consistent and conservative, ensuring this requires access to the underlying solver's shape functions $N(x)$ and is, thus, not suited for black-box coupling [50]. For each quantity, it needs to be decided, whether the mapping should be consistent or conservative. This question needs to be decided by the user, taking the overlying type of multi-physics problem into account.

A more in-depth discussion of that topic, including the conservation of virtual work over interfaces can be found in [10, 50].

2.2 Interpolation Methods

Various methods exist for interpolation that are used in the context of multi-physics simulations. As laid out by [50], the original choice of mapping methods implemented in preCICE is based on the comparisons done in [10, 11]. Relevant criteria are conservation and consistency properties, accuracy, computational efficiency and - last but not least - compatibility with the black-box approach. I will give a short overview over the mapping methods commonly found in the fluid-structure interaction context. This list in no way aims to be exhaustive.

2.2.1 Nearest-Neighbor

The nearest-neighbor interpolation method, also known as proximal or nearest-element interpolation is based on finding the geometrically nearest neighbor, i.e., the vertex with

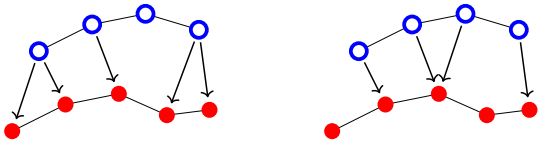


Figure 2.1: Nearest-neighbor mapping from blue (empty circles) to red (filled circles). The consistent mapping looks for a nearest neighbor for every vertex in the output mesh (red) whereas conservative mapping looks for a neighbor for every vertex in the input mesh (blue). As a result, for the consistent mapping, values from the input mesh potentially end up unmapped. For the conservative mapping, vertices on the output mesh may not receive a value or receive multiple values, which are summed up.

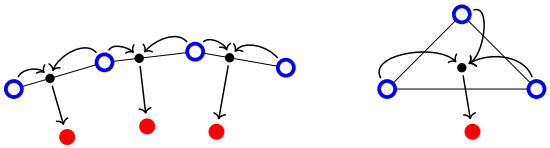
the shortest distance from the target or source vertex. Usually, the Euclidean norm is used for distance calculations. The value at the nearest neighbor of an output vertex is copied to the latter, resulting in first order accuracy. Benefits of this methods are its easy implementation and the absence of an interpolation matrix, thus dramatically reducing memory consumption. It does not take any information of the mesh beyond vertex positions into account. After the indices of the nearest neighbors of the output vertices have been computed, the method acts purely on local data and is therefore embarrassingly parallel. To derive a conservative mapping from this consistent formulation, we transpose the mapping matrix (see also Fig. 2.1), which may result in a mapping, where some output vertices do not receive a value, whereas at others, several values from the input mesh may be accumulated and summed up.

2.2.2 Nearest-Projection

The nearest-projection method can be regarded as an extension to the nearest-neighbor mapping. Because this method works on mesh elements, such as edges, triangles or quads, a suitable triangulation of the input mesh has to be provided by the solver.

It is based on an orthogonal projection from the output vertex on the nearest element of the input mesh. For this nearest element, the barycentric coordinates of the projection result are computed. For a triangle, the barycentric coordinates α, β and γ of a point $\boldsymbol{d}$ are computed from the coordinates of the triangle nodes $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$:

$$\boldsymbol{d} = \alpha\,\boldsymbol{a} + \beta\,\boldsymbol{b} + \gamma\,\boldsymbol{c} \tag{2.14}$$



a) b)

Figure 2.2: a) illustrates how a nearest-projection mapping first interpolates from the blue vertices to the orthogonal projection of an output vertex on the edge (small black circles) and after that copies the result to the output vertices (red circles). b) shows the interpolation scheme for a triangle.

and normalized according to

$$\alpha + \beta + \gamma = 1. \tag{2.15}$$

The output vertex value is eventually computed as the linearly interpolated value at the projection point using the barycentric coordinates as interpolation weights.

Figure 2.2 shows the mapping procedure with edges and triangles as mesh elements of the input mesh.

A more in depth explanation of the implemented algorithm can be found in [17, 50]. The barycentric interpolation is second order accurate, the mapping from the projection point to the output vertex is first order accurate. Based on the assumption that the gap between source and target meshes of the interpolation is small, compared to the mesh width, the nearest-projection interpolation is second order accurate in practice.

2.2.3 Radial-Basis Functions Interpolation

The interpolation with radial-basis functions works on scattered data and requires no additional information beyond the coordinates of the mesh nodes. The interpolator is defined as a weighted sum of radially symmetric functions placed at the input mesh's nodes. This method is presented in Sec. 2.3 in depth.

The three methods above, namely nearest-neighbor, nearest-projection and radial basis function interpolation are implemented in preCICE.

2.2.4 Polynomial Interpolation

This interpolation method works by finding a polynomial $p \in \mathbb{P}^n$ of degree n or lower that goes through the $n+1$ distinct support points (ξ, f_ξ) with $\xi \in \Xi$ the set of data sites and

$f_\xi = f(\xi)$ of an unknown function f that is to be interpolated. The interpolation condition reads

$$p(\xi) = f_\xi \quad \forall \xi \in \Xi. \tag{2.16}$$

Given the case of $n+1$ distinct support points, such a polynomial always exists and is unique [82].

Besides the representation as the linear combination of the standard polynomial basis $\boldsymbol{\gamma}$

$$p = \sum_{k=0}^{n} a_k \gamma_k \tag{2.17}$$

with coefficients $\boldsymbol{a}$, a number of different representations exist. From the standard basis and interpolation condition, a linear system arises, similar to Eq. (2.32). Its dense interpolation matrix is called *Vandermonde-Matrix*. Solving the system results in an infeasible high computational complexity of $\mathcal{O}\left(n^3\right)$. Other polynomial representations exist that can mitigate the computational complexity, such as Barycentric Lagrange interpolation or the Newton basis.

However, any of the different representations of the interpolating polynomial produces high-order polynomials when applied to real-world problems. This gives rise to extreme ill-conditioning of the problem, causing the problem to become unstable. If the locations ξ can be chosen freely, the roots of the Chebyshev polynomials can be used as such. This yields an optimal interpolation and minimises the unstable behavior [18]. In black-box applications, the interpolation points are supplied by the solvers. This makes interpolation by Chebyshev polynomials and other methods that require a special distribution of points impracticable here.

Instead of using the entire mesh to form an interpolant, one might split up the mesh into chunks and build localized interpolants by piecewise interpolation on each chunk, also called spline interpolation. This requires a triangulation of the data sites, either implicitly given by means of a regular grid or explicitly by the solvers. Especially in more than two dimensions, the triangulation is very difficult to provide. Furthermore, a degree of smoothness when the polynomials are joined together has to be given, which becomes challenging even in two dimensions [20, p. 41]. These properties result in a considerable set-up work for piecewise interpolation, especially compared to the nearly effortless (in this regard) radial-basis function interpolation.

2.2.5 (Localized) Lagrangian Multiplier Methods

The method of Lagrange multipliers was originally developed for particle and celestial mechanics, where constrained bodies are directly connected by interaction forces. The problem is first formulated as if no interaction forces exist. The interface constraints are then multiplied by an indeterminate coefficient and added to the virtual work of the systems [78]. These coefficients are called Lagrange multipliers. This corresponds to an *interface potential* connecting different partitions, such as for the fluid domain:

$$\pi_f = \int_\Gamma \lambda_f (u_f - u_b)\, \mathrm{d}\Gamma, \tag{2.18}$$

where Γ denotes an interface, u_b and u_f the frame displacements on each side of the interface and λ_f the Lagrange multiplier. In the general case, upon discretization, the nodes of Γ match neither of the interface meshes. As a result, the integral is carried out twice for each side of Γ. The potential π is then enforced as a boundary condition on the energy functional of the respective partitions. A localized version of the formulation can be derived by using Dirac delta functions.

For matching meshes, the construction of the intermediate mesh Γ is straightforward, for non-matching meshes however this is non-trivial, but allows to enforce conservation of a physical quantity.

As it becomes obvious from the formulation above, the interpolation needs access to the energy functional, which is usually part of the inner implementation of solvers. This requirement renders the method not applicable to black-box coupling.

2.2.6 Mortar Methods

The Mortar method can also be viewed as a global Lagrange multiplier method, having a single Lagrange multiplier field [87]. These methods [8] have become a popular choice in the case the shape function $N(x)$ is known [105, p. 18].

Starting from the from the weak form of the coupling conditions Eq. (2.1), given as

$$\int_\Gamma \psi(\boldsymbol{x}) \big(u_f(\boldsymbol{x}) - u_s(\boldsymbol{x})\big)\, \mathrm{d}\Gamma = 0 \tag{2.19}$$

with $\psi(\boldsymbol{x})$ the vector of test functions. Following the same principles as given in Sec. 2.1

above, it can be discretized.

$$\int_\Gamma \psi(\boldsymbol{x}) N_f^T(\boldsymbol{x}) \,\mathrm{d}\Gamma\, \bar{u}_f = \int_\Gamma \psi(\boldsymbol{x}) N_s^T(\boldsymbol{x}) \,\mathrm{d}\Gamma\, \bar{u}_s \tag{2.20}$$

The standard Mortar method is based on the weak-form Eq. (2.20) and uses the shape functions also as test functions $\psi(\boldsymbol{x}) \coloneqq N_f(\boldsymbol{x})$:

$$\int_{\Gamma_f} N_f(\boldsymbol{x}) N_f^T(\boldsymbol{x}) \,\mathrm{d}\Gamma_f\, U_f = \int_{\Gamma_{s\to f}} N_f(\boldsymbol{x}) N_s^T(\boldsymbol{x}) \,\mathrm{d}\Gamma_{s\to f}\, U_s, \tag{2.21}$$

where $\Gamma_{s\to f}$ is the overlap area between the fluid mesh and the projections of the structure elements. To be able to evaluate $N_f(\boldsymbol{x}) N_s^T(\boldsymbol{x})$ on $\Gamma_{s\to f}$, the structure elements are projected from Γ_s to $\Gamma_{s\to f}$, forming M_{fs}. It can be rewritten into matrix-vector form

$$M_{ff}\, U_f = M_{fs}\, U_s \quad \Rightarrow \quad H_{fs} = M_{fs}^{-1}\, M_{sf}. \tag{2.22}$$

M_{ff} can be obtained from Eq. (2.6), while M_{fs} must be constructed on the newly created mesh $\Gamma_{s\to f}$.

Enhancements, such as the Dual Mortar methods, presented in [111], render the mapping highly efficient.

Still, it is immediately eminent, that the Mortar method requires access to the shape functions $N(\boldsymbol{x})$ and is therefore not suitable for black-box coupling.

2.3 Interpolation with Radial-Basis Functions

The goal of any interpolation is to find a transformation from a set of distinct values to a continuous function or, to put it another way, to obtain values between the given ones. This section will give an introduction to the mathematical theory of interpolation by means of radial-basis functions. It follows [19, 20] in structure and nomenclature.

Given data in d dimensions, consisting of data sites $\xi \in \mathbb{R}^d$ and associated values $f_\xi \in \mathbb{R}$, we seek an approximant $s : \mathbb{R}^d \to \mathbb{R}$. The values are assumed to stem from a function $f : \mathbb{R}^d \to \mathbb{R}$, with $f_\xi = f(\xi)$. The function f itself is unknown or at least unavailable for a sufficiently large number of evaluations. The following analytical results, however, assume that f is given in closed form, e.g., for the purpose of comparing f and s for approximation quality measurements. Typical error estimates also require some smoothness of f.

The data sites can usually be restricted to a domain $\Omega \subset \mathbb{R}^d$, where Ω is prescribed

from the overarching problem, e.g., a simulation domain. In this case, one seeks an approximation $s : \Omega \to \mathbb{R}$.

Here, the approximation will take place by way of interpolation, i.e., $s(\xi) = f_\xi$, $\forall\, \xi \in \Xi$, with $\Xi \subset \mathbb{R}^d$ denoting the set of data sites. This is called the interpolation condition. Nonetheless, other choices, such as least squares approximation or "quasi-interpolation" are possible. Here, I will focus on the *exact* interpolation condition.

If the underlying data received is multi-dimensional, i.e., stems from a vector-valued function $F : \mathbb{R}^d \to \mathbb{R}^m$, $m > 1$, the approximation is performed component wise.

For interpolation with radial-basis functions, the approximant s is a linear combination of translated basis functions $\varphi(\|\cdot\|)$ where $\|\cdot\|$ is the Euclidean norm. Applying the norm gives the method a certain insensitivity to the dimensionality of the problem domain. Radially symmetric means that the function value solely depends on the distance from the origin and is invariant to rotations. The functions are translated by the data sites in Ξ. That also means that Ξ has a two-fold impact on the approximation, first as support points of the interpolation problem, second as the translation vector for the basis functions $\varphi(\|\cdot - \xi\|)$. These translations are called centres and make the approximation space S depending on the set of data sites Ξ:

$$S = \left\{ \sum_{\xi \in \Xi} \lambda_\xi \, \varphi(\|\cdot - \xi\|) \,\middle|\, \lambda_\xi \in \mathbb{R} \right\}. \tag{2.23}$$

Together with the general definition of the basis function

$$\varphi(\|\cdot - \xi\|) : \mathbb{R}_+ \to \mathbb{R}, \quad \xi \in \mathbb{R}, \tag{2.24}$$

the standard interpolant can be given as

$$s(x) = \sum_{\xi \in \Xi} \lambda_\xi \, \varphi(\|x - \xi\|), \quad x \in \mathbb{R}^d, \, \lambda_\xi \in \mathbb{R}. \tag{2.25}$$

Table 2.1 and Fig. 2.3 show a selection of commonly used basis functions. Their choice among others determines important properties, such as convergence rate, accuracy and parallelizability.

Together with the interpolation condition

$$s(\xi) = f_\xi, \quad \forall \xi \in \Xi, \tag{2.26}$$

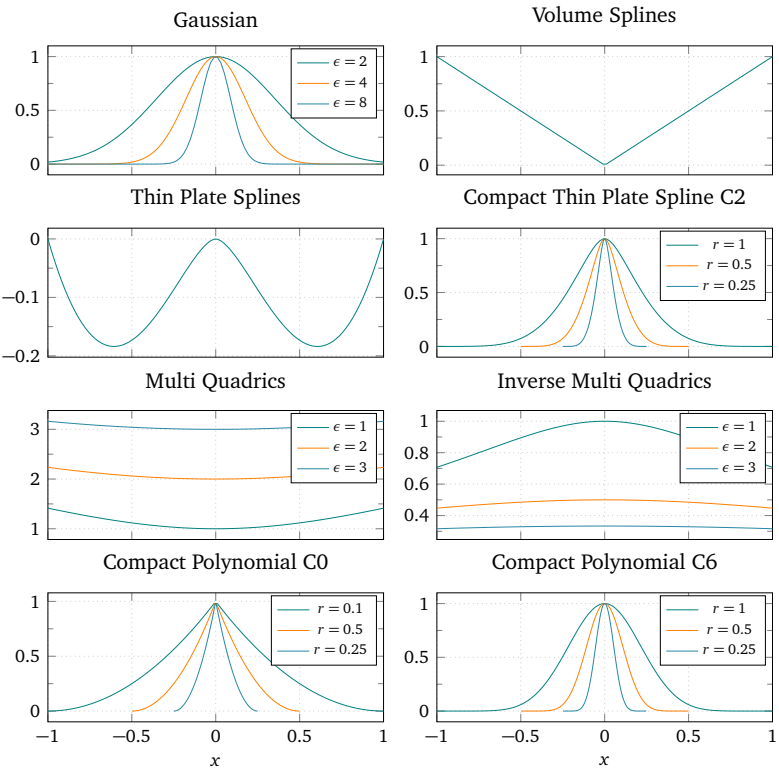


Figure 2.3: Selection of radial-basis functions available in preCICE. For the definition and meaning of the parameters ϵ and r, see Tab. 2.1

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

Table 2.1: Types of standard radial-basis functions implemented in preCICE for data mapping between non-matching meshes. ϵ is the so-called shape-parameter, $\|x\|$ the Euclidean distance of the evaluation point from the center of the basis function. For basis functions with local support, the support radius is given by r, i.e., $\varphi(\|x\|) = 0$ for $\|x\| > r$ and $\tilde{r} = \|x\|/r$ denotes the normalized distance from the center.

Eq. (2.25) can be expressed as a system of linear equations

$$C \cdot \boldsymbol{\lambda} = \boldsymbol{f} \tag{2.27}$$

with

$$C = \{\varphi(\|\xi - \zeta\|)\}_{\xi,\zeta \in \Xi} \tag{2.28}$$

$$\boldsymbol{\lambda} = (\lambda_\xi)_{\xi \in \Xi} \tag{2.29}$$

$$\boldsymbol{f} = (f_\xi)_{\xi \in \Xi} \tag{2.30}$$

from which the weights $\boldsymbol{\lambda}$ can be computed.

C in many cases is positive definite, in such cases the radial-basis function is called positive definite, as well. The Gaussian basis function with a positive shape parameter ϵ is an example of a positive definite basis function. Another one is the inverse multiquadric function. On the other hand, functions such as thin-plate splines are only conditionally positive definite of some order k [19]. To be uniquely solvable, the linear system that arises from these functions, require augmentation by a polynomial of degree $k-1$, see Sec. 2.3.1 below.

Due to the definition of the interpolation matrix and the radial symmetry of the basis function, we also know that C is symmetric.

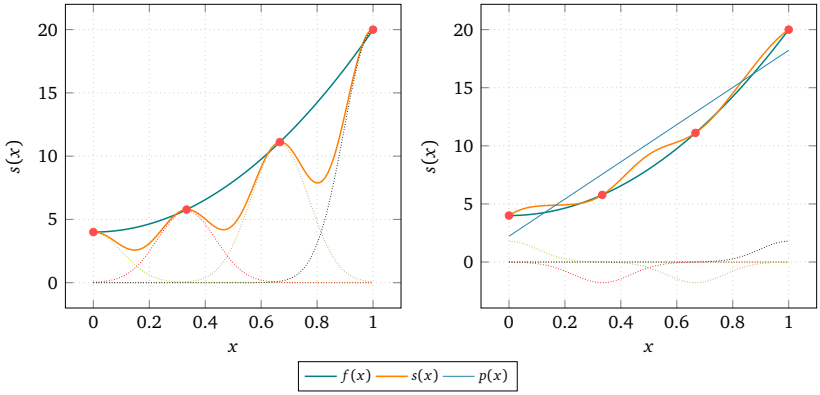


Figure 2.4: Comparison of the interpolation of $f(x) = 16x^2 + 4$ with and without linear polynomial. Interpolation points are marked and the shifted and scaled Gaussian basis functions with a shape parameter of ≈ 6.83 are shown.

For the evaluation of the interpolants at the output mesh, I define the set $\Delta \subset \mathbb{R}^d$ of target sites. The basis functions, translated by ξ and evaluated at output data sites δ, can be written in matrix-form

$$A = \{\varphi(\|\xi - \delta\|)\}_{\xi \in \Xi,\, \delta \in \Delta} \tag{2.31}$$

and then be applied to obtain the interpolated values s_δ

$$s_\delta = A \cdot \boldsymbol{\lambda} = A C^{-1} \cdot \boldsymbol{f}. \tag{2.32}$$

In practice, C will usually not be inverted explicitly, but $\boldsymbol{\lambda}$ will be determined by applying an iterative solver.

2.3.1 Polynomial Extension

To improve accuracy and condition of the interpolation, it can be helpful to add a polynomial $p(x) \in \mathbb{P}_d^{k-1}$ of degree $k-1$ in d unknowns to the right-hand side of Eq. (2.25).

In the case of a basis function which is only conditionally positive definite, the polynomial also renders the problem uniquely solvable again. Thin-plate splines basis functions are such a case of a conditionally positive definite basis function [19], where the polynomial

even is necessary to ensure a unique solution. The approximant consequently becomes

$$s(x) = \sum_{\xi \in \Xi} \lambda_\xi \, \varphi(\|x - \xi\|) + p(x). \tag{2.33}$$

This, obviously, introduces extra degrees of freedom for the coefficients of the polynomial. We restrict to the commonly used linear polynomial $p \in \mathbb{P}_d^1$. Accordingly, we add additional conditions by requiring the coefficient vector $\boldsymbol{\lambda}$ to be orthogonal to the polynomial space $\mathbb{P}_d^1(\Xi)$, which denotes the polynomials, restricted to the centers $\xi = \left(\xi^{(i)}\right)_{i=1\ldots d}$ in Ξ:

$$\sum_{\xi \in \Xi} \lambda_\xi \, q(\xi) = 0, \quad \forall q \in \mathbb{P}_d^1 \tag{2.34}$$

$$\Leftrightarrow \quad \sum_{\xi \in \Xi} \lambda_\xi \, \xi^{(i)} = 0, \quad \forall i = 1, \ldots, d \quad \text{and} \quad \sum_{\xi \in \Xi} \lambda_\xi = 0 \tag{2.35}$$

which are then enforced alongside the interpolation conditions Eq. (2.26). For a unique solution of the polynomial to exist, I require P_d^{k-1}-unisolvency on Ξ. For $k = 2$ and $d \geq 2$, for instance, this means that Ξ must not be a subset of a straight line. Furthermore, for solvability, $k \leq \|\Xi\|$ is required [90]. I take both conditions for granted in the following.

Even in cases, where this is not needed from an analytical point of view, the polynomial helps to improve accuracy, especially for certain distributions of points in the input or output meshes. This is illustrated by the obvious ability to reproduce polynomials of degree one exactly. Figure 2.4 shows this on a simple example.

Later on, in Sec. 3.2, I will discuss different strategies to treat the polynomial extension in the solution algorithm as well results for polynomials of higher degrees.

2.3.2 Convergence and Stability Properties

Convergence and stability are among the most important properties of an interpolation scheme or algorithm. In this section, I will present analytical results from literature. These results will be verified and extended later by using numerical techniques.

Let h be the mesh width, or more exactly, in cases of anisotropic meshes, the largest width between two adjacent nodes, defined by

$$h = h_{\max} := \sup_{x \in \Omega} \min_{1 \leq i \leq |\Xi|} \|x - \xi_i\|_2, \tag{2.36}$$

also called the *fill distance* or *mesh norm*. It can be seen as the radius of the largest sphere completely contained in Ω that contains no data sites. $h_{\max}$ is important for understanding

approximation and convergence orders.

Furthermore, the *separation distance* q is defined as

$$q := \min_{\xi,\zeta\in\Xi,\ \xi\neq\zeta} \|\xi-\zeta\|_2 \tag{2.37}$$

and gives the smallest distance between the two closest nodes. For a uniform mesh, the mesh width and separation distance are equal, i.e., $h = q$.

For basis functions of thin-plate spline type, convergence results have been published in [19]. Given φ is a basis function of such type in d dimensions, defined by

$$\varphi(r) = \begin{cases} r^{2k-d}\log r, & \text{if } 2k-d \text{ is an even integer,} \\ r^{2k-d}, & \text{if } 2k-d \text{ is not an even integer,} \end{cases} \tag{2.38}$$

and s the standard interpolant with polynomial Eq. (2.33).

Then, according to [19, Theorem 2], there exists an h-independent constant c such that,

$$\|s-f\|_{\infty,\Omega} \leq c \begin{cases} h\sqrt{\log(h^{-1})}, & \text{if } 2k-d=2, \\ \sqrt{h}, & \text{if } 2k-d=1 \text{ and } 0\leq h\leq 1, \\ h, & \text{in all other cases .} \end{cases} \tag{2.39}$$

As mentioned above, in practice a linear polynomial is used, thus $k = 2$. Given the two-dimensional case, $d = 2$, thin-plate splines as described in Tab. 2.1 obviously fit this type of function. Multiquadrics can be derived from this type by composition with $\sqrt{a^2+r^2}$ and $\varphi(r)$ in three dimensions.

Still, for other basis functions used in applications, such as thin-plate splines in three dimensions or Gaussians, the restriction on certain classes of functions and dimensionalities makes the aforementioned proof not applicable.

Geometry wise, the results above hold for a very general, bounded domain Ω, given it contains the aforementioned $\mathbb{P}_d^{k-1}$-unisolvent subset, making the polynomial unique on that domain.

Further results are available for functions to be approximated $f \in H^{2k}(\Omega)$ in the Sobolev spaces. On infinite grids ($\Xi = h\,\mathbb{Z}^d$) an optimal uniform convergence rate can be given as $\mathcal{O}\left(h^{2k}\right)$.

A number of convergences proofs require that is f is a function in the so-called *native space*. Given a positive distributional Fourier transform $\hat{\varphi}(\|\cdot\|) : \mathbb{R}^d \setminus \{0\} \to \mathbb{R}$ of the basis

function φ and $\hat{f}$ as the Fourier transform of f, the square of the native space norm is

$$\|f\|_{\varphi}^2 := \frac{1}{(2\pi)^d} \int \frac{1}{\hat{\varphi}(\|t\|)} \left|\hat{f}(t)\right|^2 \mathrm{d}t \tag{2.40}$$

and the native space X is the space of all f on $\mathbb{R}^n$ for which Eq. (2.40) is finite. The native space, belonging to the basis function φ, can be seen as defined by all distributions f that make this particular seminorm finite [19]. A more comprehensive introduction and survey concerning native spaces can be found in [91].

For practical use cases, the requirement of $f \in X$ seriously restricts the practical usefulness of proofs that are based on it. Many analytic functions are excluded from X and it is difficult to characterize which of them are contained in X. Also, for some basis functions, notably Gaussians, the native spaces are quite small [79]. Moreover, in a black-box scenario, an analytic function can not be given at all. I will still give literature results that use this requirement and compare them later to results retrieved from experiments.

For Gaussians, exponential convergence rates can be given, under the condition of $f \in X$

$$\|s - f\|_{\infty,\Omega} \leq \exp\left(-c \frac{\log h}{h}\right) \|f\|_X \tag{2.41}$$

and also for inverse quadrics

$$\|s - f\|_{\infty,\Omega} \leq \exp\left(-c \frac{1}{h}\right) \|f\|_X . \tag{2.42}$$

A similar error bound also holds for multiquadrics [79].

Requiring $f \in X$ also allows to refine the result from Eq. (2.39) for $k = d = 2$. Then, c can be expressed as $c = \tilde{c} \|f\|_{\varphi}$, where $\tilde{c}$ only depends on Ω [19].

In [79, 80] the general convergence behavior for RBF interpolation with focus on Gaussian basis functions is analyzed. Platte compares the convergence to the best polynomial approximation. The Lebesgue constant compares the approximation quality of s to the best polynomial approximation (not necessarily interpolation) [58, p. 47]. It is defined as

$$\|f - s\| \leq (\Lambda_n + 1) \|f - p^*\| , \tag{2.43}$$

where p^* is defined as

$$p^* = \arg\min_{p \in \mathbb{P}^n} \|f - p\| , \tag{2.44}$$

the best polynomial approximation of degree n. Therefore the interpolation s is at most a

factor $\Lambda_n + 1$ worse than the best possible polynomial interpolation of a given degree.

In the following I will use the L_∞-norm. $\Lambda = \|s\|_{\text{op}}$ will then be the maximum L_1-norm of a row of S [102].

An exponential growth of Λ indicates unstable interpolation behavior. This unstability may stem from the existence of a Runge phenomenon and in fact the Lebesgue constant provides a very well measure of the Runge phenomenon [47, s. 6.1.1] when used together with the L_∞-norm.

In [79] the general convergence behavior for Gaussian und inverse quadrics basis functions with varying shape parameter is reviewed. Platte compares RBF interpolation with polynomial interpolation and again uses the Lebesgue constant to define unstable behavior. He states for equally spaced nodes, while polynomial and Gaussian RBF interpolation significantly differ, this difference decreases with diminishing shape parameter ϵ and increasing number of nodes N, more precicely as $\epsilon^2/N \to 0$. For interpolation with Gaussian RBFs, he distinguishes between three regimes of interpolation behavior, depending on the shape parameter choice.

For $\epsilon \to 0$, *the flat limit*, the RBF method with smooth basis functions becomes identical to polynomial interpolation and unstable, exhibiting a Runge phenomenon. A proof is given in [34].

With a fixed shape parameter, the RBF interpolation convergences asymptotically, as $N \to \infty$, similarly to Lagrange polynomial interpolation and yields unstable behavior according to the definition above using the Lebesgue constant. Also, if polynomial interpolation fails to converge to a given target function, then RBF interpolation will also fail.

Of particular interest is the case of $\epsilon = \mathcal{O}\left(\sqrt{N}\right)$. This case exhibits exponential convergence behavior for sufficiently smooth functions but due to the existence of Runge regions, the process is still unstable.

As a last case [79] mentions the stationary case, i.e. $\epsilon = \mathcal{O}(n)$, where the interpolant fails to converge due to saturation error, but it leads to better conditioned interpolation matrices and a stable interpolation process. Given a constant size of the domain Ω, the Gaussian basis function scales the same way as the mesh density, therefore the term *stationary*. This scaling was also used by me to define the shape parameter independently from the mesh width in Eq. (3.2).

In general, RBF interpolation is susceptible to the Runge phenomenon and large Lebesgue constants for both equidistant as well as Gauss-Chebyshev distributed nodes, albeit to a lesser extend in the latter case. As with polynomial interpolation, the rapid growth of Λ_n can be prevented by selecting best nodes for interpolation.

2.3.3 Error Saturation

Aside from the questions of convergence and stability, touched in the previous section Sec. 2.3.2, another accuracy limiting phenomenon exists. Even as h approaches zero, the error may not converge to zero, but may have a lower bound e_s, the *saturation error*. Error saturation exists for many types of basis functions, including Gaussians, as reviewed in [43, p. 130]. The saturation error is a function of the width of the basisfunctions relative to the mesh width. Therefore, to properly analyze the saturation error, the shape parameter ϵ is scaled with the mesh width h. This is called the *stationary case*, as the basis functions remains stationary compared to the mesh. ϵ is chosen inversely proportional to the mesh width or fill distance, such that

$$\epsilon = \frac{\alpha}{h}. \tag{2.45}$$

Note that, m from Eq. (3.2) and α are interchangeable by $\alpha = \sqrt{-\ln 10^{-9}}/m$. The Gaussian basis function is then given as

$$\varphi(x) = \exp\left(-\alpha^2 \left(\frac{x}{h}\right)^2\right). \tag{2.46}$$

For infinite, uniform grids and Gaussian basis functions, Boyd and Wang [73] and Maz'ya and Schmidt [16] have shown, that the saturation error is

$$\|s - f\|_\infty = e_s(\alpha) = 4 \exp\left(-\frac{\pi^2}{\alpha^2}\right) \cdot \|f\|_\infty \tag{2.47}$$

for a smooth function f and its RBF interpolant s.

On infinite, non-uniform grids, the situation becomes worse, as shown in [14] and the saturation error rises with

$$\|s - f\|_\infty = e_s(\alpha) = c \cdot \exp\left(-\frac{\pi^2}{4\alpha^2}\right) \cdot \|f\|_\infty . \tag{2.48}$$

However, no analytical results on the practical case of finite grids are available. The analytical results for infinite grids are assumed to be a lower bound for finite grid cases.

In [12], Boyd uses a combination of analytical and numerical methods to reach results on the behavior of the saturation error on finite grids for Gaussian basis functions. He gives an approximation to the saturation error on finite, uniform grids as

$$e_s \approx 0.058 \exp\left(-\frac{0.467}{\alpha^2}\right), \tag{2.49}$$

which, notably, has the same form as the analytical results on infinite grids, i.e., $e_s \sim \exp(1/\alpha^2)$. The result is obtained by interpolating the smoothest possible function, $f(x) \equiv 1$ to provide a practical lower bound for the saturation number, compared to more realistic and rougher functions.

Obviously, the saturation error can be arbitrarily small if choosing a sufficiently small α, i.e., large basis functions. However, this causes the condition number of the matrix C to increase until it becomes an accuracy limiting factor, replacing one problem by another. For infinite grids, Boyd and Wang concludes that $\alpha \approx 0.5$ is a safe operating value, as the saturation error is below machine epsilon and the condition number of the interpolation matrix still reasonably low. This corresponds to a value of $m \approx 9.1$.

The results above give the order of a lower bound to the saturation error on uniform or infinite grids, respectively. For general, irregular grids and different test functions, the saturation error will only be larger.

2.4 Solution Approaches for RBF problems

It is a well-known problem of RBFs to produce ill-conditioned linear systems, described in [13] as “a major curse for RBF methods”. Figure 2.5 illustrates that. The condition as well as accuracy increase together with a growing support radius (or flatness) of the Gaussian basis functions. The algorithm eventually reaches a point, where the rounding errors due to the ill-conditioning do not allow the accuracy to improve further, indicated by oscillations in both. The existing unique solution of the linear system may, therefore, never been found. In the figure, the shape parameter ϵ is controlled by the parameter m which normalizes it to mesh width, see Eq. (3.2).

According to [45], errors in typical RBF implementations grow with the number of nodes for quasi-uniform distributions for two reasons. The first reason is the increased condition number. The second reason is an intrinsic ill-conditioning for a quasi-uniform point distribution leading to large errors near the boundaries.

[47] showed that the eigenvalues of the matrix C follow a distinct pattern where the

magnitude of the eigenvalues scales with powers of the shape parameter ϵ:

$$\begin{aligned}
&\{\mathcal{O}(1)\},\\
&\{\mathcal{O}(\epsilon^2), \mathcal{O}(\epsilon^2)\},\\
&\{\mathcal{O}(\epsilon^4), \mathcal{O}(\epsilon^4), \mathcal{O}(\epsilon^4)\},\\
&\{\mathcal{O}(\epsilon^6), \mathcal{O}(\epsilon^6), \mathcal{O}(\epsilon^6), \mathcal{O}(\epsilon^6)\},\\
&\dots
\end{aligned} \tag{2.50}$$

until the last eigenvalue is reached. The kind of pattern depends on the dimensionality of input data and whether it is arranged on a sphere. It holds true for inverse quadrics, multiquadrics and Gaussian basis functions, though for (inverse) multiquadrics Fornberg and Zuev uses a slightly different definition, reading

$$\varphi(x) = \sqrt{1 + (\epsilon r)^2}, \tag{2.51}$$

for multiquadrics and analog for inverse multiquadrics.

The pattern Eq. (2.50) is valid for two-dimensional data on non-periodic meshes, for other kinds of input problems, similar patterns can be found. Furthermore, the scattering of the input points does not play a role.

It is obvious from that pattern, that the problem of ill-conditioning becomes worse with

- larger problems, since then eigenvalues proportional to higher powers of the shape parameter come into play, and
- smaller shape parameters, since the condition grows with the smallest eigenvalue shrinking with $\mathcal{O}(\epsilon^n)$ according to the foregoing pattern.

As a result, the entries of RBF coefficient vector $\boldsymbol{\lambda}$ become very large (some positive, some negative), causing a lot of numerical cancellation to happen. The final result is therefore obtained by two potentially ill-conditioned steps, Eqs. (2.25) and (2.31). The values $s(\delta), \delta \in \Delta$ to be obtained, however, generally depends in a well-conditioned way on the data $\{\xi, f_\xi\}$.

In [15], Boyd and Gildersleeve provide experimental results for the growth of the condition number κ for several basis functions in the case of sufficiently large and uniform grids. For Gaussians in one-dimensional space, they get

$$\kappa = \frac{1}{2} \exp\left(\left(\frac{\pi}{4\epsilon h}\right)^2\right) \tag{2.52}$$

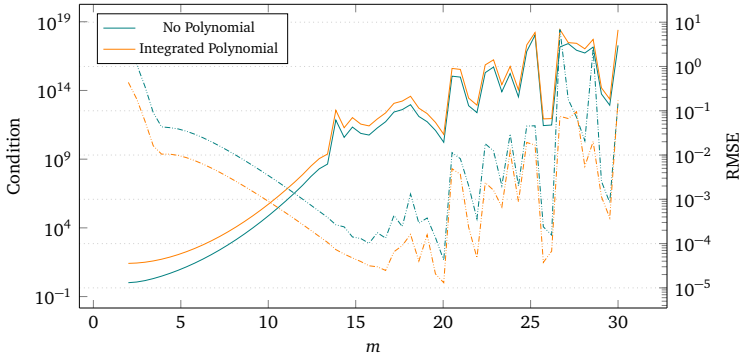


Figure 2.5: Condition (solid line) and root mean squared error (dashed line) for Gaussian basis function on $[0, 1]$ with 500 support points for increasing support width.

and

$$\kappa = \frac{1}{4} \exp\left(\left(\frac{\pi}{2\epsilon h}\right)^2\right) \tag{2.53}$$

for Gaussians in two-dimensional space. Furthermore, results are given for inverse multiquadrics

$$\kappa = \frac{1}{2} \exp\left(\frac{\pi}{\epsilon h}\right), \tag{2.54}$$

and

$$\kappa = 0.363\, n^2 \exp\left(\frac{3.07}{\epsilon h}\right) \tag{2.55}$$

for multiquadrics, both in one-dimensional space. For thin-plate splines, which do not have a shape parameter,

$$\kappa = 0.1198139997\, n^4 \tag{2.56}$$

could be obtained. While all of the results were obtained from numerical experiments, only for the first two they were able to match them to known numbers (such as π here). For non-uniform meshes κ generally is larger, possibly by orders of magnitude.

The exponential growth of the condition number with $1/\epsilon$ very well illustrates the numerical challenges one needs to overcome when using larger support radii to achieve higher accuracy. Adjusting ϵ allows to control this trade-off between stability and accuracy, but will always be a compromise and does not yield optimal accuracy.

In the following subsections I will review general approaches to circumvent the problem of ill-conditioning to increase numerical stability and/or accuracy. The methods will be

reviewed in terms of their compatibility with the black box coupling implemented in preCICE. Methods which are suitable in terms of parallelizability and are realizable in a black-box coupling environment will be evaluated further in Chapter 4.

2.4.1 Preconditioning

Besides the standard preconditioning techniques for linear systems, there exist methods specifically tailored for RBF interpolation.

One example is preconditioning by accelerated iterated approximate moving least squares, presented in [42]. The technique has shown good first results reducing the condition number on uniform meshes. However, achieving stability on non-uniform meshes remains problematic, as the method requires control of the distance of the interpolation system matrix to the identity matrix. In the core of the method is the truncated Neumann series $\sum_{k=0}^{n}(I-A)^k$ that converges to A^{-1} if and only if $\|I-A\| < 1$. Thus, the method is not fully applicable to coupled problems with black-box solvers where the eigenvalues of the interpolation matrix are not known a priori.

2.4.2 Basis Transformations

The idea of this group of methods is to replace the basis based of translates on radial basis functions by a basis that spans the same function space, but lessens the ill-conditioning and therefore is less prone to numerical instability.

[29] presents first results of a basis transformation, also based on the concept of native spaces. However, their research was still in the early stages and requires a suitable set of support points, such as trigonometric Gauss or Gauss-Legendre points to support a stable multi-dimensional quadrature. Furthermore, the basis arises from a weighted singular value decomposition, which is required for the kernel matrix C.

In [45], Fornberg, Larsson, and Flyer propose a basis that does not show ill-conditioning with decreasing shape parameter ϵ. Based on earlier research in [46], which requires the input points to be located on a sphere, they were able to generalize the method to up to three dimensions. The basic idea of the method is to express the original vector of RBF evaluations $\mathbf{\Phi}$ as

$$\mathbf{\Phi}(\boldsymbol{x}) = \varphi \|x-\xi\|_{\xi\in\Xi} \tag{2.57}$$

$$= K\,E\,\boldsymbol{y}(\boldsymbol{x}) \tag{2.58}$$

where K is a matrix with entries of size $\mathcal{O}(1)$, $\boldsymbol{y}$ is a vector of spherical harmonics and $E = \mathrm{diag}\{\epsilon^0, \epsilon^2, \epsilon^4, \ldots\}$ a diagonal matrix of increasing powers of ϵ. As an example, it is shown here for the two-dimensional, non-periodic case. The pattern of powers of ϵ stems from an expansion of the basis function φ and matches the pattern given in Eq. (2.50). The scaling of the basis function has therefore been confined to the matrix E.

By multiplying from the left and splitting up $K = QR$ using QR-decomposition, a new basis Ψ can be created while the space that is spanned by Φ remains the same.

$$\boldsymbol{\Psi}(\boldsymbol{x}) = E_N^{-1} Q^T \boldsymbol{\Phi}(\boldsymbol{x}) = E_N^{-1} Q^T K E \boldsymbol{y}(\boldsymbol{x}) = E_N^{-1} R E \boldsymbol{y}(\boldsymbol{x}) \tag{2.59}$$

where E_k denotes the first $k \times k$ part of E. The expression $E_N^{-1} R E$ is well-conditioned and upper triangular.

However, there are a number of downsides of this method. To construct the decomposition shown in Eq. (2.59), one needs an expansion of the basis function into this particular structure. This needs to be done separately for each function and dimensionality. So far, the authors were only able to find such an expansion for Gaussian and Bessel type basis functions in up to three-dimensional space. Furthermore, Eq. (2.59) requires a QR-decomposition of the dense matrix C. While this only needs to be done in the initialization or offline-phase of the algorithm, it is potentially very costly in terms of computation and communication.

The method has been implemented as part of [97] and will be further scrutinized in Sec. 4.6.

2.4.3 Finding an Optimal Shape Parameter

Some basis functions have a shape parameter, such as Gaussians or (inverse) multiquadrics. It is usually treated as a parameter to the algorithm, i.e., it must be set manually. Especially for Gaussians the shape parameter is of importance since it controls the size of the function's support and has direct influence on the sparsity patterns of the resulting matrices C and A, as well as condition and accuracy. Accuracy losses both occur in the high ϵ-regime, due to lacking basis function support in parts of the domain, as well as for low values of ϵ due to severe ill-conditioning. Most of the guidelines given in literature are trying to find a suitable balance between these adverse effects.

It is common practice to include at least the nearest neighbor [11, 31] in both the input and the output mesh. For a non-uniform mesh, the largest distance should be considered [10]. While these statements are based on experiences, [41] gives a systematic approach

to this issue, based on *leave-one-out cross validation* (LOOCV). Whereas naive LOOCV requires to create one interpolant for each vertex followed by an optimization step, [85] was able to simplify the algorithm such that the interpolant needs to be generated only once for all vertices. Because it is still followed by the optimization step to determine the shape parameter and needs to explicitly create the inverse of the interpolation matrix, it significantly increases the computational cost. The algorithm also optimises only with respect to accuracy and does not take the harming effect of the ill-conditioning into account. Therefore, I use a heuristic approach to determine the best value for m taking the guidelines given above into account.

2.4.4 Adaptive Techniques

A number of different approaches exist to how adaptiveness can be achieved for a RBF implementation. The basis function used in the algorithm is considered fixed, but can be parameterized with the shape parameter ϵ. Together with the set of input data sites Ξ, ϵ determines the algorithm's behavior.

The simplest method of "adaptiveness" is to calculate a global shape parameter from the mesh width as given in Eq. (2.36). Especially for Gaussians, a "good" relation between h and ϵ can be found by means of setting the width of the basis function in terms of h, using Eq. (3.2). This is a global method and only yields good results on uniform meshes. There are methods to find an optimal shape parameter systematically, see Sec. 2.4.3.

An extension is to introduce a shape parameter ϵ_i for each basis function, based on a local measure of mesh density. This way, a compactly supported basis function can be scaled so that it always includes the m nearest-neighbors. A support radius including a fixed number of vertices directly reflects in the sparsity pattern of the interpolation matrix C, resulting in $2m+1$ non-zero entries per row. On the other hand, an adaptive support radius breaks the symmetry of the matrix and it can not be guaranteed to be non-singular anymore. Given the scattered data mapping constraints, a local mesh density measure can not be assumed to exist and needs to be computed beforehand. To explore the obvious advantages for non-uniform meshes, I will still evaluate a locally adaptive shape parameter in Sec. 4.5.

Many adaptive methods require that the target function f can be evaluated at any time. In [35] a residual sub-sampling method is used. The interpolant is created on a uniform grid which is subsequently refined at intermediate points placed halfway between centers. If the error exceeds a certain threshold, additional points are inserted. Driscoll and Heryudono also adapt the shape parameter to be constant with respect to the local mesh

density, sharing the effects mentioned in the previous paragraph. Still, the most prominent drawback is that f must be evaluable at runtime.

Zhang, Zhao, and Levesley pursue in [112] a similar idea of adding and removing data sites based on a pointwise error estimator. The main difference to the aforementioned method is that no additional function evaluations are required. Instead, the error indicator is based on the idea that two different interpolation methods will give significantly different results in regions challenging for the interpolations. For each point ξ an neighboring set of points is created from which an interpolant is constructed. The difference from this interpolator to the global interpolator on the working set of points will be used as an error indicator and points will be added (refined) or removed (coarsened) from the working set accordingly. The method is described only for multiquadrics basis function with a variable shape parameter which is scaled reciprocal to the distance to the nearest neighbor. In principle, the method, however, should work with any basis function.

All the aforementioned methods require knowledge about the local mesh density to create new points in a neighborhood of existing ones or to control the support radius of the basis functions. Computing this information purely from scattered data can not only be infeasible expensive but might also lead to unexpected results in highly anisotropic meshes. Furthermore, runtime modification of the mesh and evaluation of f at arbitrary locations require communication with the respective solvers and would violate the black-box coupling principle.

2.4.5 Multiscale Interpolation

Wendland describes a multilevel approximation method in [110]. In contrast to the two aforementioned methods, the basic version does not adaptively refine or coarsen the existing mesh, but instead uses a sequence of meshes $\Xi_1, \Xi_2, \ldots, \Xi_n$ that can be created a priori. However, extensions to multiscale interpolation, that do an adaptive discarding of points are also presented below. The idea resembles the multigrid linear solver method, in how it builds the interpolant on the residual, rather than the target values.

On each of the meshes Ξ_j an interpolation space can be built

$$W_j = \operatorname{span}\{\varphi_j(\cdot, \boldsymbol{x}) : \boldsymbol{x} \in \Xi_j\}. \tag{2.60}$$

The final approximation of the function f comes from the sum of these spaces

$$V_n := W_1 + W_2 + \ldots + W_n \tag{2.61}$$

for $n \to \infty$. Algorithm 1 gives the basic idea of the algorithm.

Data: f_ξ, Number of levels n
Result: Approximate solution $g_n \in W_1 + \ldots + W_n$
$g_0 \leftarrow 0$, $e_0 \leftarrow f_\xi$
for $j = 1, \ldots, n$ **do**
  Determine the local interpolant $s_j \in W_j$ to e_{j-1}
  $g_j \leftarrow g_{j-1} + s_j$
  $e_j \leftarrow e_{j-1} - s_j$
end

Algorithm 1: Basic multiscale interpolation

The basis function φ_j on Ξ_j, which is used in constructing the interpolant s_j is scaled, so that the support radius is r_j. Given a basis function $\varphi : \mathbb{R}^d \to \mathbb{R}$ with support in the unit sphere, the scaled version can be defined as

$$\varphi_j(\boldsymbol{x}, \boldsymbol{y}) \coloneqq r_j^{-d}\, \varphi\left(\frac{\boldsymbol{x} - \boldsymbol{y}}{r_j}\right). \tag{2.62}$$

The scaling and the refinement is controlled by two critical parameters $\mu \in (0, 1)$ and ν. The first parameter μ controls the refinement of the sequence of meshes between subsequent levels, so that the mesh width at level $j+1$ is $h_{j+1} \approx \mu h_j$. This ensures a certain uniformity in decrease. Analogously, ν controls the decrease of the support radius $r_{j+1} \approx \nu r_j$. The parameters allow to control the size and the number of non-zeros of the interpolation matrix at each level and have therefore direct influence on the computational cost. For simplicity, one can set $\nu = \mu^{-1}$.

Wendland proposes two variations of the algorithm, which both work by discarding certain information. The first one, originally introduced in [69], discards all coefficients $|\lambda| \leq \tau$ of the interpolant with an absolute value smaller then τ, as given in Alg. 2. For the meaning of λ, see Eq. (2.25). The threshold τ should depend on the level j.

The second variation uses adaptive refinement of the sequence of meshes. It computes the error $e_j = e_{j-1} - s_j$ on the upcoming mesh X_{j+1} and will discard those points from X_{j+1} where the absolute error is below a level-depended threshold $\boldsymbol{\kappa}$, see Alg. 3.

As it becomes evident from the algorithms given above, the method requires a number of parameters set, namely μ, ν and $\boldsymbol{\tau}$ or $\boldsymbol{\kappa}$, respectively. Wendland uses a value of $\mu = 0.5$ for the refinement factor and ν in the range of $3 \ldots 11$. The support radius is then controlled by setting $r_i = \nu q_j$, based on the separation distance q, Eq. (2.37).

Data: f_ξ, Number of levels n, Threshold $\boldsymbol{\tau}$
Result: Approximate solution $g_n \in W_1 + \ldots + W_n$
$\tilde{g}_0 \leftarrow 0$, $\tilde{e}_0 \leftarrow f_\xi$
for $j = 1, \ldots, n$ **do**
  Determine the local interpolant $s_j \in W_j$ to $\tilde{e_{j-1}}$
  Drop all coefficients $|\lambda_k^{(j)}| \leq \tau_j$ in s_j to define $\tilde{s}_j$
  $\tilde{g}_j \leftarrow \tilde{g}_{j-1} + \tilde{s}_j$
  $e_j \leftarrow e_{j-1} - \tilde{s}_j$
end

Algorithm 2: Multiscale interpolation with dynamical discarding of interpolation coefficients smaller then τ.

Data: f_ξ, Number of levels n, Thresholds $\boldsymbol{\kappa}$, Meshes $\tilde{\Xi}_1, \ldots, \tilde{\Xi}_n$
Result: Approximate solution $g_n \in W_1 + \ldots + W_n$
$g_0 \leftarrow 0$, $e_0 \leftarrow f_\xi$, $\Xi_1 \leftarrow \tilde{\Xi}_1$
for $j = 1, \ldots, n$ **do**
  Determine the local interpolant $s_j = I_{X_j, \varphi_j} e_{j-1} \in W_j$
  $g_j \leftarrow g_{j-1} + s_j$
  $e_j \leftarrow e_{j-1} - s_j$
  for $\boldsymbol{x} \in \tilde{\Xi}_{j+1}$ **do**
    if $|e_j(\boldsymbol{x})| > \kappa_j$ **then**
      $\Xi_{j+1} = \Xi_{j+1} \cup \{\boldsymbol{x}\}$
    end
  end
end

Algorithm 3: Multiscale interpolation with adaptive discarding of points

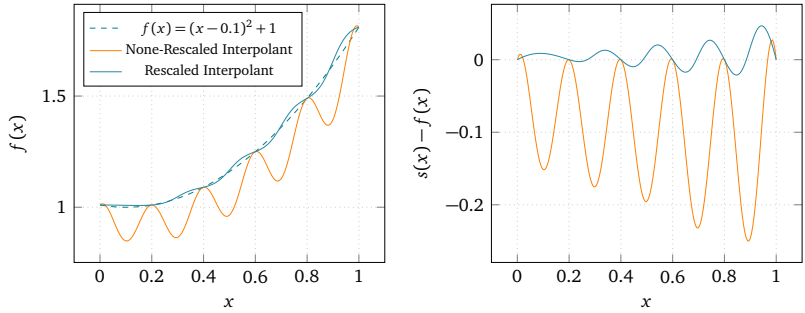


Figure 2.6: Demonstration of insufficient width or density of basis functions and how rescaling can mitigate the effect. Gaussian basis functions with a shape parameter of $\epsilon = 9$ are used.

As with other methods mentioned in this section, the most significant drawback is the requirement of a sequence of increasingly finer meshes. These meshes need to be created manually beforehand or algorithmically, which in turn requires neighborship information. While this information can be assembled in $\mathcal{O}(|\Xi_n| \cdot \log|\Xi_n|)$ using tree-based algorithms (see also Sec. 3.4), it still requires a coarsening strategy and can lead to unexpected results in heavily distorted meshes.

2.4.6 Rescaled Interpolation

In [31] a rescaling procedure is proposed. The method does not try to directly address the ill-conditioning, but improves interpolation quality. This makes it possible to use basis functions with a smaller support radius, which, in turn, improves the condition number of the interpolation problem as well as its parallelizability. The method shows very promising results during first investigations and is further evaluated in Sec. 4.4. For a thorough theoretical underpinning of the method, see [28]. Basically, it works by constructing a rescaled interpolant s_r by

$$s_r(\boldsymbol{x}) = \frac{s_f(\boldsymbol{x})}{s_1(\boldsymbol{x})} \tag{2.63}$$

where $s_f(\boldsymbol{x})$ denotes the interpolant constructed from the given values as shown in Eq. (2.25) and $s_1(\boldsymbol{x})$ is the interpolant of the constant function $g(\boldsymbol{x}) = 1$.

The rescaling addresses an issue with localized, e.g. Gaussian, basis function which decrease monotonically with increasing distance to their centers. Figure 2.6 demonstrates

the effect, dubbed *hanging laundry* effect. When the local density of support points is too low, the interpolant sags between two neighboring support points.

Substantial improvements can be observed in particular for small m without negatively affecting the conditioning. The additional computational cost remains moderate as it only involves the computation of one additional interpolant for each set of data positions to be mapped. Furthermore, parts of the computation can be reused for solving the interpolation system later.

3 Optimization & Implementation of RBFs in preCICE

> The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming.
>
> Donald E. Knuth

In this section, some special aspects of RBF interpolation as realized in preCICE are highlighted further.

The mapping algorithm is implemented within preCICE in two different versions. A basic, non-parallel version uses a direct QR-solver. The Eigen [55] linear algebra library is used for matrix storage and solving. This version does not implement features such as using a separated polynomial.

The more advanced version, on which I will focus here, is based on the PETSc suite. *PETSc, pronounced PET-see (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism. PETSc (sometimes called PETSc/Tao) also contains the Tao optimization software library.* (from Balay et al. [4]). Components used in this implementation are the matrix and vector libraries as well as application orderings (AO), Krylov subspace (KSP) and QR-decomposition based solvers.

Specifics of the implementation are detailed here, including re-enumeration of vertices by using a mapping between an application and a PETSc specific ordering, employing preallocation for fast matrix assembly and the usage of a spatial index for fast finding of vertices inside a specific sphere. Furthermore, the implementation of the different basis functions as well as how the matrix expressions are evaluated, are laid out in this section.

3.1 Basis Functions

The basis function $\varphi : \mathbb{R}^+ \to \mathbb{R}$ forms the kernel of the method. As mentioned before, its choice among others determines important properties of the algorithm, such as convergence rate, accuracy and parallelizability. Table 2.1 shows a number of standard basis functions implemented in preCICE.

Global, but monotonically decreasing basis functions, such as Gaussians can be transformed to local functions by introducing a cut-off radius r and shifting them to achieve continuous functions:

$$\varphi_{\text{local}}(x) = \begin{cases} \varphi_{\text{global}}(x) - \varphi_{\text{global}}(r) & \text{for } x < r, \\ 0 & \text{for } x \geq r \end{cases}. \tag{3.1}$$

This ensures that the interpolation matrix C is sparse, even for global basis functions, given a sufficiently small cut-off radius. Obviously, a meaningful cut-off radius can only be given, when $\varphi(x) \to 0$ as $x \to \infty$.

Some basis functions have a so-called shape parameter ϵ. Its effect on the basis function and thereby on the whole interpolation depends on the type of basis function. For Gaussians it controls the slenderness of the function, while for multiquadrics the effect is more that of a vertical shift and with inverse multiquadrics it mostly effects the value at $x = 0$, see Fig. 2.3 for the effect of the shape parameters on φ. The concrete ramifications of different basis functions and values for the shape parameter are further detailed in Chapter 4.

In the following, I use a fixed cut-off value of $r = 10^{-9}$, i.e., return a value of 0 for $\varphi(x) < 10^{-9}$. This cut-off procedure is especially suited for Gaussians, since from an analytical perspective they never reach 0, albeit from a numerically perspective they fall under machine precision[1] when $x > 6.06\sqrt{\epsilon^{-1}}$ and below the cut-off when $x > 4.55\sqrt{\epsilon^{-1}}$.

For some basis functions, in particular Gaussians, the support radius can be expressed as multiples m of the mesh width, i.e., $r = m \cdot h_{\text{max}}$. h_{max} is the maximum mesh width of the input mesh, as defined in Eq. (2.36). Based on the definition of the Gaussian basis function, the corresponding shape parameter ϵ can then given by

$$\epsilon = \frac{\sqrt{-\ln 10^{-9}}}{m \cdot h_{\text{max}}} \tag{3.2}$$

This ensures that the basis function decays to 10^{-9} within the m neighboring vertices from

[1] which is 2^{-52} for a 64 bit double

its center if the distance between data points is $h_{\max}$.

For the other compact basis functions, *Compact Thin Plate Splines C2*, *Compact Polynomial C0* and *Compact Polynomial C6*, according to their definitions, see Tab. 2.1, the shape parameter immediately determines the support radius. Thus, the shape parameter for these three basis function is simply given by

$$\epsilon = m \cdot h_{\max}. \tag{3.3}$$

A suitable choice for the shape parameter ϵ is of tremendous importance for the resulting approximation quality and convergence properties. This can be illustrated by $\epsilon \to 0$, an infinitely flat basis function. The resulting matrix would only consist of ones and thus be singular, which corresponds to condition number ∞. On the other hand, $\epsilon \to \infty$ yields singular peaks as basis functions, resulting in a diagonal system matrix but very poor interpolation quality.

In preCICE, the basis function is implemented as a class which is given the support radius and shape parameter ϵ at instantiation time, if applicable. For Gaussians, the effective cut-off radius r is then set as $r = \min\{\text{supportRadius}, \sqrt{-\log 10^{-9}/\epsilon}\}$ and henceforth the basis function is evaluated as given in Eq. (3.1).

3.2 Evaluation of the Polynomial

3.2.1 Consistent Interpolation

Starting from the standard interpolant with polynomial, Eq. (2.33), there are different variants of evaluating Eq. (2.32) together with the polynomial, affecting both stability and accuracy of the results as well as load-balancing in parallel scenarios.

The polynomial can be integrated by augmenting matrices C and A, yielding the extended coefficient vector $\tilde{\boldsymbol{\lambda}}$ and the vector $\boldsymbol{s}_\delta$ of values at the evaluation points:

$$\underbrace{\begin{pmatrix} 0 & Q^T \\ Q & C \end{pmatrix}}_{\tilde{C}} \underbrace{\begin{pmatrix} \beta \\ \lambda \end{pmatrix}}_{\tilde{\lambda}} = \underbrace{\begin{pmatrix} 0 \\ f_\xi \end{pmatrix}}_{\tilde{f}_\xi} \quad \Rightarrow \quad \tilde{\boldsymbol{\lambda}} = \tilde{C}^{-1} \cdot \tilde{\boldsymbol{f}}_{\tilde{\xi}}, \tag{3.4}$$

$$s_\delta = \underbrace{\left(\begin{array}{c|c} V & A \end{array}\right)}_{\tilde{A}} \cdot \tilde{\boldsymbol{\lambda}} = \tilde{A}\tilde{C}^{-1} \cdot \tilde{\boldsymbol{f}}_\xi. \tag{3.5}$$

where Q and V hold the coordinates of the input or output mesh, respectively, defined as $Q = (1\ \xi_i^{(1)} \dots\ \xi_i^{(d)}) \in \mathbb{R}^{|\Xi| \times (d+1)}$ and $V = (1\ \delta_i^{(1)} \dots\ \delta_i^{(d)}) \in \mathbb{R}^{|\Delta| \times (d+1)}$.

$\boldsymbol{\beta} \in \mathbb{R}^{d+1}$ holds the corresponding coefficients of the polynomial.

The polynomial can also be solved separately from the core RBF interpolation. This involves solving for the coefficients $\boldsymbol{\beta}$, subtracting the polynomial from the input vector $\boldsymbol{f}_\xi$ and finally adding the polynomial to the output vector, as outlined below:

1. Solve the over-determined system using a QR decomposition $Q = \bar{Q}\bar{R}$. This yields the least-squares solution for the coefficients of the polynomial $\boldsymbol{\beta}$, according to

$$\left[\begin{array}{c} Q \end{array}\right] \cdot \left[\boldsymbol{\beta}\right] \approx \left[\begin{array}{c} \boldsymbol{f}_\xi \end{array}\right] \quad \Rightarrow \quad \boldsymbol{\beta} = \bar{R}^{-1}\bar{Q}^T \boldsymbol{f}_\xi = Q^+ \cdot \boldsymbol{f}_\xi. \tag{3.6}$$

   Q^+ denotes the pseudo-inverse $\bar{R}^{-1}\bar{Q}^T$ of Q.

2. Subtract the polynomial $p(\xi)$ from the input vector, i.e., solve

$$\begin{aligned} & C \cdot \boldsymbol{\lambda} = \boldsymbol{f}_\xi - Q \cdot \boldsymbol{\beta} \\ \Leftrightarrow \quad & \boldsymbol{\lambda} = C^{-1}\left(\boldsymbol{f}_\xi - Q \cdot \boldsymbol{\beta}\right) \\ & \quad = C^{-1}\left(\boldsymbol{f}_\xi - QQ^+ \cdot \boldsymbol{f}_\xi\right). \end{aligned} \tag{3.7}$$

3. Add $p(\delta) = V \cdot \boldsymbol{\beta}$, evaluated at the output sites to the output vector, i.e.,

$$\begin{aligned} \boldsymbol{s}_\delta &= A \cdot \boldsymbol{\lambda} + V \cdot \boldsymbol{\beta} \\ &= A \cdot C^{-1}\left(\boldsymbol{f}_\xi - QQ^+ \cdot \boldsymbol{f}_\xi\right) + VQ^+ \cdot \boldsymbol{f}_\xi \end{aligned} \tag{3.8}$$

3.2.2 Conservative Interpolation

Equation (2.32) gives the consistent formulation of the interpolant. From that, the conservative interpolation can be derived by transposing the interpolation operator:

$$\boldsymbol{s}_{\delta} = \left(A C^{-1}\right)^{T} \cdot \boldsymbol{f}_{\xi} = C^{-1} A^{T} \cdot \boldsymbol{f}_{\xi}. \tag{3.9}$$

Based on Eq. (3.9), the conservative interpolation with integrated polynomial can be constructed similarly to the consistent version, outlined above:

$$\boldsymbol{s}_{\delta} = \tilde{C}^{-1} \tilde{A}^{T} \cdot \tilde{\boldsymbol{f}}_{\xi}. \tag{3.10}$$

The version with separated polynomial for conservative interpolation can be constructed by transposing Eq. (3.8).

$$s_{\delta} = H \cdot \boldsymbol{f}_{\xi} = \underbrace{\left[A C^{-1}\left(I - Q Q^{+}\right) + V Q^{+}\right]^{T}}_{H} \cdot \boldsymbol{f}_{\xi} \tag{3.11}$$

$$= \left[\underbrace{C^{-1} A^{T}}_{2} - Q^{+T}\left(\underbrace{Q^{T} \underbrace{C^{-1} A^{T}}_{3}}_{} - \underbrace{V^{T}}_{1}\right)\right] \cdot \boldsymbol{f}_{\xi}. \tag{3.12}$$

The actual evaluation of Eq. (3.11) is done in multiple steps, as indicated by the underbraced numbers and detailed in the following steps:

1.
$$\boldsymbol{\omega} := V^{T} \cdot \boldsymbol{f}_{\xi} \tag{3.13}$$

2. Compute $C^{-1} A^{T} \boldsymbol{f}_{\xi}$. This involves solving the linear system C.

$$\boldsymbol{\eta} := A^{T} \cdot \boldsymbol{f}_{\xi} \tag{3.14}$$

$$C \cdot \boldsymbol{\mu} = \eta \quad \Rightarrow \quad \boldsymbol{\mu} := C^{-1} \cdot \boldsymbol{\eta} = C^{-1} A^{T} \cdot \boldsymbol{f}_{\xi}. \tag{3.15}$$

3.
$$\boldsymbol{\tau} := Q^{T} \cdot \boldsymbol{\mu} - \boldsymbol{\omega} \tag{3.16}$$

4.
$$\boldsymbol{\sigma} := Q^{+T} \cdot \boldsymbol{\tau} \tag{3.17}$$

Again, Q^{+} denotes the pseudo-inverse of Q. The QR-decomposition yields $Q = \bar{Q}\bar{R}$

and thus:

$$\boldsymbol{\sigma} = \left(\bar{Q}\,\bar{R}^{-T}\right) \cdot \boldsymbol{\tau}. \tag{3.18}$$

5. The result can finally be given as:

$$\boldsymbol{s}_\delta = \boldsymbol{\mu} - \boldsymbol{\sigma}.$$

The evaluation requires one solve of a linear system, one QR-decomposition of the tall-and-skinny matrix Q and four matrix-vector products per interpolation step.

As mentioned above, Q and V hold the coordinates of the input or output vertices, respectively, and therefore are both densely populated matrices. In cases of compactly supported basis functions, C and A are sparse matrices and, given an appropriate ordering of vertices, even diagonally dominant. These favourable properties are destroyed or at least impaired for the integrated polynomial as shown in Eqs. (3.4) and (3.9).

While the integrated and separated formulation are mathematically not equal, they yield almost identical results. In Sec. 4.3, both variants will be compared for both numerical stability/condition as well as equivalence of results.

3.3 Rescaling of the Interpolant

The rescaling of the interpolant as presented in Sec. 2.4.6 and in [28, 31] is straightforward to implement. The coefficients $\boldsymbol{\lambda}_1$ for the constant right-hand side $\boldsymbol{g}_\xi = 1$ can be computed once in the offline phase as

$$\boldsymbol{\lambda}_1 = C^{-1} \cdot \boldsymbol{g}_\xi. \tag{3.19}$$

Thus, for each interpolation in the online phase, we only have to calculate

$$\boldsymbol{s}_1 \coloneqq A\boldsymbol{\lambda}_1 \quad \text{and} \tag{3.20}$$

$$\boldsymbol{s}_\delta \coloneqq \frac{\boldsymbol{s}_\delta}{\boldsymbol{s}_1}, \tag{3.21}$$

where the fraction above is interpreted as *pointwise* division of two vectors. Occasionally, entries of value zero may occur in $\boldsymbol{s}_1$, which therefore must be checked for prior to division. This happens, when the basis function's support radius is chosen too small for the input mesh and points are not supported by at least one basis function. In this case, however, not only the rescaling procedure is impaired, but the mapping as a whole will suffer from severe accuracy degradation.

The rescaling is currently only implemented for consistent interpolation with separated polynomial. For the integrated polynomial, the matrix $\tilde{C}$ from Eq. (3.4) yields a perfect approximation of g_ξ, as the polynomial alone gives a perfect fit. Using C without the integrated polynomial doesn't allow reusing of the matrix C as well as the associated solver and hence requires to duplicate the creation and solving for the interpolation matrix. These steps constitute for a significant amount of computational effort of the offline phase. Furthermore, the presence of the polynomial makes s_δ not exhibiting the same *hanging laundry* effect (compare Sec. 2.4.6) as the corrector s_1, which is thus not suited for correcting it.

Using the rescaling method for conservative interpolation, regardless of the type of polynomial, is also not feasible, since it would break conservativness of the interpolation.

3.4 Algorithmic & Implementation Overview

The implementation of a basic RBF algorithm is easy and straightforward. Most of the complexity stems from the setup involving different parallel solvers, from optimizations and from the fact that we hand over the system to PETSc as a solver library.

Two stages can be identified, usually called *online* and *offline*. The latter must be executed only at the beginning of the simulation or - more precisely - whenever the locations of support or evaluation points change. The principal steps of the algorithm are:

1. Creation of the mapping between preCICE and PETSc vertex indices, as described later in this section.
2. Preallocation of the system matrix C and evaluation matrix A, see Sec. 3.4.4.
3. Filling of these matrices, together with Q and V, if applicable. See Alg. 4.
4. Initialization of the linear solvers for matrices C and Q.
5. Computation of the rescaling coefficients by solving the linear system C for $g_\xi = 1$.
6. Filling of evaluation vectors, solving of the system and copying results back into preCICE data structures.

Steps 1 through 5 are part of the offline phase while step 6 needs to be performed on every coupling iteration. Step 5 is mostly equivalent to step 6 for the specific right-hand side of $g_\xi = 1$.

Data: InputMesh, OutputMesh, Basis Function φ
Result: C, A, Q, V

if *conservativeMapping* **then**
  $\Xi \leftarrow$ OutputMesh
  $\Delta \leftarrow$ InputMesh
else if *consistentMapping* **then**
  $\Xi \leftarrow$ InputMesh
  $\Delta \leftarrow$ OutputMesh
end
for $i = 1, \ldots, |\Xi|$ **do**
  if ξ_i *not owned by current rank* **then** continue
  if *polynomial* = *Integrated* **then** $C_{\cdot,i+d+1} \leftarrow (1\ \xi_i^{(1)} \ldots\ \xi_i^{(d)})^T$
  if *polynomial* = *Separated* **then** $Q_{i,\cdot} \leftarrow (1\ \xi_i^{(1)} \ldots\ \xi_i^{(d)})$
  for $j, \ldots, |\Xi|$ **do**
    if *supportRadius* $< \left\|\xi_i - \xi_j\right\|$ **then**
      $k, l \leftarrow$ AOApplicationToPetsc(i, j)
      $C_{k,l} \leftarrow \varphi(\left\|\xi_i - \xi_j\right\|)$
    end
  end
end
for $i = 1, \ldots, |\Delta|$ **do**
  if *polynomial* = *Integrated* **then** $A_{i,\cdot} \leftarrow (1\ \delta_i^{(1)} \ldots\ \delta_i^{(d)})$
  if *polynomial* = *Separated* **then** $V_{i,\cdot} \leftarrow (1\ \delta_i^{(1)} \ldots\ \delta_i^{(d)})$
  for $j, \ldots, |\Xi|$ **do**
    if *supportRadius* $< \left\|\delta_i - \xi_j\right\|$ **then**
      $l \leftarrow$ AOApplicationToPetsc(j)
      $A_{i,l} \leftarrow \varphi(\left\|\delta_i - \xi_j\right\|)$
    end
  end
end

Algorithm 4: Filling of the matrices, including the mapping from application ordering using preCICE global indices to PETSc ordering. The preallocation as well as the treatment of dead dimensions is omitted here for the sake of brevity.

3.4.1 Dimensionality of Input Data and Dead Dimensions

The spatial dimensionality of the input data coordinates is given at runtime to the RBF implementation. While the RBF method is dimension-agnostic by using the L_2-norm on the argument of the basis function, the implementation limits the dimensionality to three.

Some solvers, notably OpenFOAM, always work on three dimensional data, even when the actual problem is of lesser dimensionality. The solver then uses an *empty* boundary condition to denote that a dimension is not used. To reflect this behavior inside preCICE, dimensions can be marked as *dead*, which causes its coordinates set to zero with respect to the interpolation. Algorithm 5 shows the treatment of dead dimensions when filling matrix C. The respective part of the algorithm for matrix A works alike.

```
1 x = ξ_i − ξ_j
2 for k = 1, ..., d do
3 |   if k is marked dead then
4 |   |   x_k ← 0
5 |   end
6 end
7 C_ij ← φ(||x||)
```

Algorithm 5: Treatment of dead dimensions when computing the entry C_{ij} inside the inner loop of the respective part of Alg. 4.

3.4.2 Re-Partitioning of Meshes

As outlined in Sec. 1.3.2, the selected mapping plays an important role in the process of re-partitioning the mesh supplied by the solvers. Again, without loss of generality, we assume that the mapping happens at participant A and the interface mesh Γ_B must therefore be made available to participant A. In step IV of the re-partitioning process, see Tab. 1.1 and Fig. 1.4, the mapping is responsible for selecting which vertices are needed on a particular slave to successfully perform the distributed mapping.

The selection of vertices required for the mapping works in a two-level tagging process, realized by two callback functions `tagMeshFirstRound` and `tagMeshSecondRound`, which are implemented as part of the mapping code. The process is illustrated in Fig. 3.1, the steps in the following refer to that illustration.

First, each slave at participant A constructs a bounding box, enlarged by the support radius of the selected basis function, around its local mesh. Then all vertices of the remote

mesh Γ_B that are contained in that bounding box (step I) are marked. Note that, if the *pre-filter/post-filter* variant is used, Γ_B was already pre-filtered in step III. From this first round marking, the owner information is derived, i.e., each vertex is *owned* by one particular rank. To guarantee uniqueness, the ownership assignment is done in a centralized fashion by the master. The information which vertices are marked is sent back to the master. Since a vertex can be and in practice is marked by multiple ranks, but can only be owned by one, the master first equally distributes vertices among ranks. The leftover vertices are subsequently assigned to the ranks that marked them, in order of ranks. While this approach does not generate a perfect load distribution, it performs sufficiently well. The owner information for each slave is then send back to it. This ensures that the evaluation matrix A can be constructed distributed among all ranks.

In the second round, step III, vertices needed for the local construction of matrix C are marked. The procedure is similar to the first round. A bounding box, enlarged by the support radius, is constructed around all owned vertices. All vertices inside that box are then marked as needed for the construction of C.

The final mesh $\tilde{\Gamma}_B^i$ at slave i of participant A is the result of the union of both rounds of tagging. It now contains all vertices needed to perform the mapping. The information which vertex is present at which rank is finally sent to the master. Thus, a re-partitioning of $\Gamma_B = \cup \tilde{\Gamma}_B^i$ has been achieved. Figure 3.1 illustrates the procedure. Note that, the support radius can be infinite in case of globally supported basis functions. In that case, the entire mesh Γ_B is present at each rank of A. In the course of the process, each vertex is assigned a *global index*, which is used to uniquely identify a vertex among all ranks and to assign it to a row in the distributed construction of the global matrices A and C, see the following sections.

The re-partitioning using the other mapping methods, i.e., nearest-projection and nearest-neighbor interpolation, work alike, albeit adapted to their specific requirements to perform a local mapping.

3.4.3 Vertex & Row Ordering

As mentioned above, preCICE uses the PETSc library to provide the functionality for distributed linear algebra. In order to leverage the performance of the built-in parallel solvers, the matrix assembly must additionally be parallelized. PETSc uses a row-wise decomposition of matrices, i.e., each complete row is assigned to one rank and each rank holds a continuous chunk of rows of a matrix or vector. The parallel layout of C and A resembles the parallel layout of the input and output meshes, i.e., each local support point

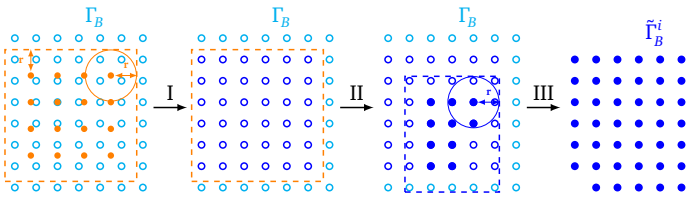


Figure 3.1: Process of the re-partitioning of meshes with RBF mapping. Compare Fig. 1.4 and Tab. 1.1 for a high-level overview. In step I, all vertices of Γ_B (light blue circles) that are contained in a bounding box around Γ_A^i (orange circles), enlarged by the support radius r are marked (blue circles). These vertices are required for constructing the evaluation matrix A. In step II, a subset of the marked vertices are assigned to the rank i and said to be owned by it (blue dots). Again, an enlarged bounding box is used to mark the vertices needed for construction of the interpolation matrix C in step III. The resulting, re-partitioned mesh $\tilde{\Gamma}_B^i$ finally contains all vertices needed to perform the mapping locally on rank i.

ξ corresponds to a local row of matrix C and each local output vertex δ corresponds to a local row of A, respectively. Filling of the matrices can not be done purely locally, as for each row or $\xi \in \Xi$ the basis function $\varphi(\|\xi - \zeta\|)$ must be evaluated for all $\zeta \in \Xi$. Even for basis functions that are only defined on a constrained support radius, some of the vertices ζ are located on a non-local rank.

From that problem setting, two requirements arise. First, non-local vertices, that are required for evaluation of the basis function of any local vertex, must be made available to the local rank. Second, in order to assign a vertex to a matrix row, a globally unique ID for each vertex is required.

preCICE provides different ways of identifying a vertex. The *id* of a vertex is set and fixed when the vertex is created during initialization. At this time, there may not yet be a connection established between the different ranks of one participant. Because of that and also for reasons of efficiency, the vertex id is simply implemented as a counter and does not guarantee uniqueness beyond the scope of one rank. However, it guarantees that the ids are continuous within each rank. This property is widely used throughout preCICE, e.g., for addressing coordinates of a vertex from a linear storage, which would not be possible using a generator for a so-called universally unique identifier (UUID) such as the library Boost UUID. The latter types of IDs are globally unique with extremely high probability and can be locally created, but cannot create continuously arranged integers. A vertex in preCICE can also be identified by its *global index*, which is globally unique among all vertices on

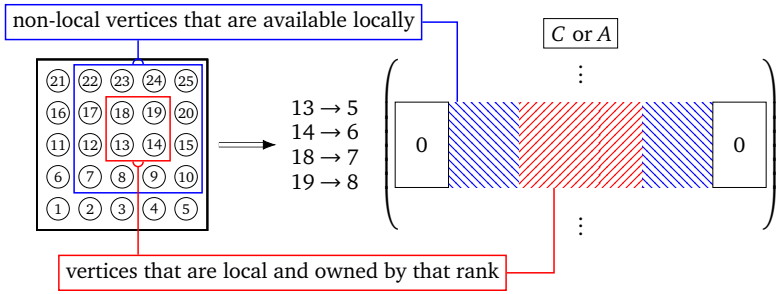


Figure 3.2: Mapping of vertices from their global index to a local position in matrix C or A using a PETSc application ordering. Here, the vertices with the *global indices* $\{13, 14, 18, 19\}$ are mapped to rows $\{5, 6, 7, 8\}$. Only the hatched part of the matrix is saved locally.

all all ranks. However, the global index is not available before the mesh re-partitioning is performed and the index is not guaranteed to be continuous among the vertices of one rank.

The re-partitioning process guarantees the validity of the global index, as aforementioned and that all vertices needed to create the local part of the mapping are present on the local rank. This includes the local part of the mesh, enlarged by the basis function's support radius, which may also be infinite. In this case, the entire input mesh Ξ has to be present to create a complete row of C or A. The entire process is explained in detail in the previous section. To distinguish these ghost vertices from the local part of the mesh, the local vertices are said to be *owned* by one rank and a corresponding property is set on each vertex. Figure 3.2 illustrates this concept. Each rank defines a local mapping by assigning all vertices it owns to continuous chunks of local rows. The size of the local portion of the matrices are given to PETSc at creation and equal the number of owned vertices, possibly plus the space needed for the polynomial on rank zero. From that local mapping, a global mapping needs to be created from the union of all local mappings.

PETSc offers so-called Application Ordering (AO) objects to facilitate a globally shared dictionary. In contrast to PETSc index sets, they allow "holes" in the indexing range, i.e., global indices that are not mapped to a row. This occurs when a vertex is filtered out by the re-partitioning process, e.g., when it is not part of the coupling interface and does not participate in the mapping. This can be the case when the vertex is either an inner vertex or the coupling interfaces of the participants do not entirely overlap. The application ordering object is created on the global MPI communicator and initialized locally with the mapping

created from the owned vertices at each rank. Indices that use the preCICE addressing scheme based on the global index can then be translated to the PETSc addressing scheme using the PETSc routine `AOApplicationToPetsc`. This guarantees that all local vertices are mapped to continuous chunks of local rows.

3.4.4 Matrix Assembly & Preallocation

The major workload during initialization of the RBF mapping is filling of the two matrices, composed of two major blocks. First, there is the cost of computing the value of each matrix entry by evaluating the basis function on each pair of vertices. Second, memory for A and C needs to be allocated, a significant factor due to the potentially large number of entries to be set in the two matrices. While both issues are separate in theory, in implementation they are tightly related. In this section, I will first show how to limit the number of necessary basis function evaluations, followed by a discussion of the issue of memory allocation. Finally, it is shown how both relate in the implementation.

In a naive implementation of the algorithm, each vertex must be subtracted from every other to form the argument of the basis function, resulting in a complexity of $\mathcal{O}\left(|\Xi|^2\right)$ for C and of $\mathcal{O}\left(|\Xi| \cdot |\Delta|\right)$ for A. Various methods are implemented to restrict the number of vertices that are to be taken into account.

For local basis functions, such as Compact Thin Plate Splines C2 and Compact Polynomial C0 and C2, a support radius r according to Tab. 2.1 can be defined. For global, but monotonically decreasing functions, it can be defined as shown in Eq. (3.1). A known support radius is only useful if the vertices within that radius can efficiently be identified. Since preCICE works on non-structured grids which not necessarily carry topological, i.e., neighborhood information, we can not rely upon information that makes fast spatial queries possible. Still, we have to avoid an exhaustive search through the entire mesh.

Spatial trees are one method to perform fast queries on multi-dimensional data. One popular choice for such an indexing structure are R-trees, proposed by Guttman in [56]. A R-tree is a height-balanced tree. Each node of the tree stores a box describing the space occupied by its children nodes. These boxes or sub-regions do not have to be disjoint, i.e., can be overlapping. Only the leaf nodes store index records, containing pointers to the data. The splitting algorithm defines when and how leaf are nodes splitted when new information needs to be integrated into the tree. Three different node splitting algorithms are available: *linear*, *quadratic* and *rstar*, each controlled by two parameters, usually called M and m. These parameters define the maximum (M) and minimum ($m \leq \frac{M}{2}$) number of index records in the leaf nodes. The resulting height of such a tree containing N index records is

at most $\log_m(N-1)$, thus, resulting in a search complexity of $\mathcal{O}\left(\log_m N\right)$. The efficiency holds not only for exact searches, but also for nearest neighbor searches as demonstrated in [25].

$(k-d-b)$-trees are another multi-level balanced tree structure, suitable for handling n-dimensional point data. In contrast to R-trees, the domain is partitioned into disjoint sub-regions. The performance for queries other than exact searches is, however, inferior to that of an R-tree [53].

The *Geometry*[2] library, which is part of the Boost project, implements spatial indices based on R-trees. Existing geometric data structures can be used by the geometry library by adapting them to so called concepts, such as a *Point*, *Box* or *Polygon*. These concepts provide an abstract model of geometric primitives. A sphere concept is, however, not supported. Relationships of these concepts to the mesh can be queried using different predicates, most notably *nearest*, *within* or the generic predicate *satisfies*. The Boost-based library was preferred over other implementations for various reasons. The software preCICE already makes heavy use of Boost libraries, so that no additional dependency is introduced. Furthermore, Boost libraries have shown a high level of code quality and long lasting support.

A mesh in preCICE is modeled as a collection of `mesh::Vertex` objects. By adapting the `mesh::Vertex` class to the point concept, preCICE vertices can be used by the library. The underlying storage container, `utils::PtrVector` is also adapted, so that it can directly be used as a container for the library and no copying of the vertices from the preCICE mesh into the tree data structures is required. Both adaptions are done by means of template specialisation, taking place at compile time. These adapters enable Boost Geometry to directly use the preCICE data structures that comprise the search space, i.e., the mesh.

The matrix assembly procedure requires to find all vertices, that lay within a support radius r of the vertex ξ in consideration. As noted above, the tree library does not offer a sphere concept, which would be a perfect match for querying points within a support radius. Hence, this type of query needs to be emulated using the existing predicates. Hereafter, different methods are evaluated in terms of performance.

naive serves as a baseline to compare the other methods to. For each query, it compares all vertices and checks if the distance is smaller then the given radius. The tree-structure is not used in this approach.

satisfy uses an anonymous λ-function, which is evaluated for every vertex ζ_i in the tree

[2] `https://www.boost.org/doc/libs/1_69_0/libs/geometry/doc/html/index.html`

and checks if it is included in the support radius of the vertex ξ, i.e., if $\|\zeta_i - \xi\| < r$. Together with the *satisfies* predicate of the library, it selects all vertices in the tree, for which the condition above holds true.

While this method uses the tree-structure, it does not use the spatial query functionality. Compared to the *naive* method, the complexity is not reduced, because the λ-function needs to be evaluated for all vertices.

boxThenDistance uses a box, that is constructed to enclose the support radius and the *within* predicate for tree-based pre-filtering of points. The preliminary resulting set of points is then filtered using the naive method.

boxAndSatisfy is logically equivalent to the previous method, but uses a logical *and* to combine both filtering steps into one expression using the enclosing box with the beforementioned satisfy predicate.

neareastThenDistance uses a tree-based k-nearest-neighbor query, followed by a pointwise distance filtering. This method requires an additional parameter k for the number of points to query for. Because the number of points inside the sphere with support radius r is unknown, a heuristic value with a safety factor that includes all points inside the sphere is required. On the other hand, k must not be too large to ensure an efficient filtering.

Each of these query methods as well as the time it took to create the R-tree are evaluated for different mesh sizes.

The heart of the R-tree algorithm is the node insertion and splitting algorithm used for creating the tree. The selection of the most efficient algorithm and parameter set for a given problem is subject to a number of factors. These include distribution of vertices inside the mesh or whether emphasis is placed on either creation or query performance, the sum of the two yielding the total cost. Here, the modification of nodes or recreation of the tree can happen only when mesh movement has taken place or a simulation restart was performed. Compared to querying for vertices, modifications of the tree are currently extremely rare events. For that reason, I put special emphasis on the query performance, rather than the creation performance.

Beckmann et al. compare in [7] existing insertion strategies and perform optimization with respect to different criteria such as the area covered or the overlap between rectangles. This optimization leads to the R*-tree insertion algorithm which is presented there. Here, I

will compare the original linear and quadratic algorithm from [56] as well as the aforementioned R*-tree algorithm. The library offers run- or compile-time selection of the algorithm. I use compile-time selection, to closely reproduce how the library will eventually be used in preCICE. For the parameters M and m, the default values of $M = 16$ and $m = 0.3 \cdot M$ are used.

The results for the initialization times can be found in Fig. 3.3 and those for query performance in Figs. 3.4 and 3.5.

The performance for finding all points inside a radius $r = 0.05$ is evaluated. While this value is chosen arbitrarily, it represents a realistic support radius. For the measurements, a test mesh consisting of n^2 grid points is set up. Vertex coordinates are constructed using the following rule:

$$P(n_x, n_y) = \begin{pmatrix} \frac{n_x}{n} + 5\frac{n_y}{n} \\ \frac{n_y}{n} + 3\frac{n_y}{n} \\ 2\frac{n_x}{n} + 2\frac{n_y}{n} \end{pmatrix} + \begin{pmatrix} 0.01 \\ 0.02 \\ -0.01 \end{pmatrix} \tag{3.22}$$

with $n_x, n_y = 0 \dots n$. This results in a two-dimensional, anisotropic mesh in three-dimensional space.

Search points are created equally distributed on the mesh, using a fixed size of $n = 100$, resulting in $n^2 = 10000$ queries. To remove direct correspondence to the index mesh, the constant shift from Eq. (3.22) is not applied. The parameter k of the *nearestThenDistance* method depends on the size of the current mesh and is set to $0.001\,n^2$. This estimate is based on the dimensions of the test mesh and the search radius r and includes a generous safety margin to reflect the uncertainty present in an actual use case.

The runs were performed on one node of the *sgscl1*[3] cluster.

Comparing the results in Fig. 3.3 with Figs. 3.4 and 3.5 shows the trade-off between creation and query performance. While all algorithms posses linear runtime complexity when it comes to creating the data structure, the R*-tree algorithm needs considerably more time building the tree. Though the performance disadvantage at creation time is confirmed by the original paper from Beckmann et al. [7], the difference observed here is much larger than in the original paper. One possible explanation for this discrepancy is the emphasis the author put on paging behavior of the algorithms. Paging is a memory management technique to swap parts of main memory to secondary memory (usually a hard disk). This helps to overcome situations where main memory size may not be sufficient for a given problem. Due to the large memory sizes of modern computers, paging only

[3] 8 Intel Xeon CPU E3-1585 v5 @ 3.50GHz, 32 GB memory

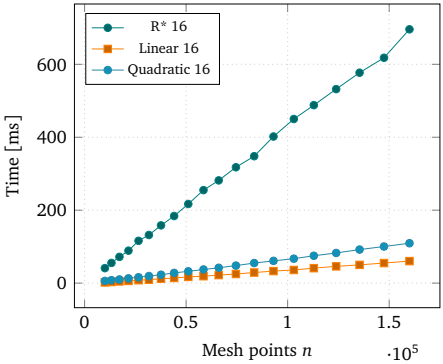


Figure 3.3: R-tree creation time for different splitting algorithms. Parameters used for the splitting algorithms are $M = 16$ and $m = 0.3\,m$. It immediately becomes evident, that the R* 16 splitting algorithm is not optimized for insertion performance. Linear and quadratic splitting algorithms enable much faster insertions, albeit at the expense of query performance.

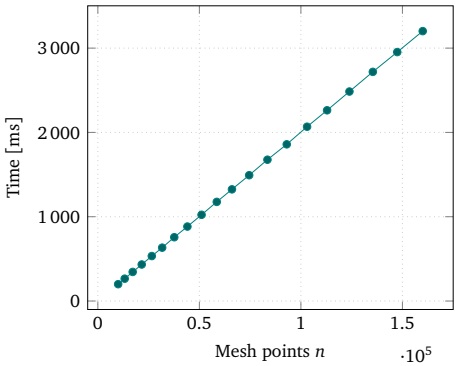


Figure 3.4: Time for 10 000 neighbor queries using the naive method on different mesh sizes.

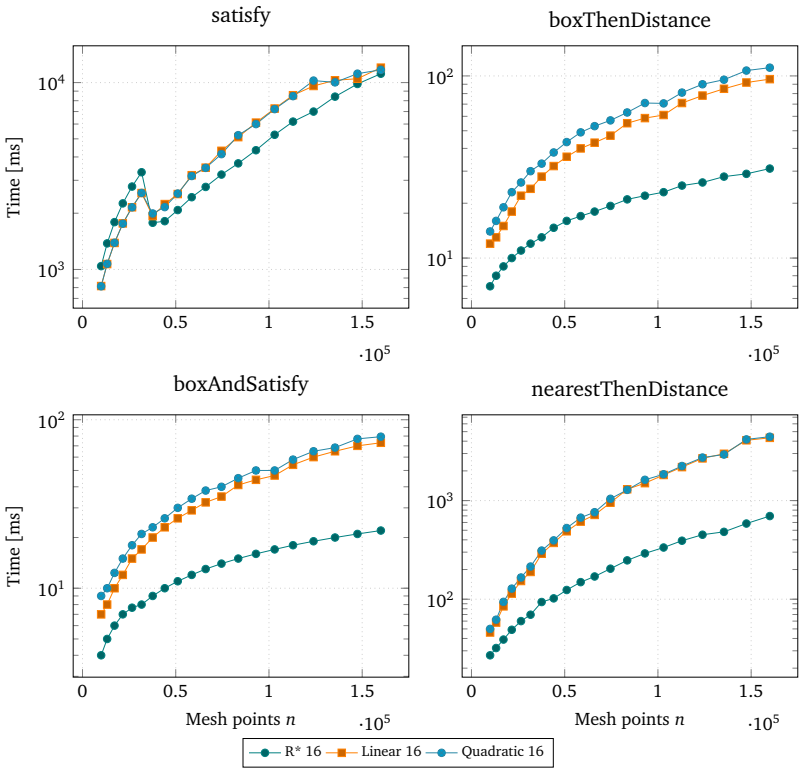


Figure 3.5: Time for 10 000 neighbor queries for points within a given support radius for different mesh sizes.

rarely occurs. Most HPC systems do not even have paging activated. Looking at query performance, the *R** algorithm performs best when compared for query performance, while *Linear* and *Quadratic* split algorithms are mostly on par.

For query performance, the naive algorithm serves as a baseline, showing almost perfect linear complexity for a constant number of queries on an increasingly large mesh. Without surprise, the *satisfy* method performs worst, even outperformed by the naive approach. It does not make use of any of the geometric concepts offered by the library and instead requires checking each vertex for the distance to the search point. With that it is logically equivalent to the naive approach. The worse runtime performance stems from the overhead of the tree structure, which is clearly not optimized for a simple traversal. *boxThenDistance* and *boxAndDistance* are not only logically equivalent, as outlined above but show also similar performance. The *nearestThenDistance* is in the middle of the field. Its main disadvantage is the additional parameter which determines the number of nearest neighbors that are initially queried for. That parameter requires assumptions about the local mesh density which is not known a priori.

The tree must be re-created each time the mesh is changed. preCICE does not support restarting at this time, so this can happen only at startup, when the mesh is given from the solvers to preCICE. The resulting tree data structure is cached at a central location, so that it can easily be re-used in different parts of preCICE. Therefore, good query performance is preferred over initialization performance and the R*-tree with *boxAndSatisfy* query method was implemented in preCICE.

In preCICE, the spatial tree is used in the mapping component for all three methods of mapping. The nearest neighbor mapping queries for one nearest neighbor of the vertex in consideration. Alike, the nearest projection mapping uses an adaption of tree indices to more complex mesh elements than vertices, i.e, edges, triangles and quads[4] and performs nearest element queries on the tree.

For the RBF mapping, the usage of the spatial tree is, however, more involved. As a first step of optimization, the spatial tree is used to limit the number of vertex pairings that need to be evaluated. This limits the set of vertices to be tested in the inner loop of Alg. 4, lines 12 and 22. In the actual implementation, this optimization is used tightly together with another optimization measure that reduces the number of necessary memory allocations and is detailed in the next section.

[4]Quad type elements remain to be implemented, as of today

Preallocation

Allocation is the act of reserving chunks of memory that are used to store values. The size of memory that is to be reserved is determined by the number of values and the respective data type. Allocation can be done individually for each datum whenever the value is computed. For matrices, this results in a number of allocations equal or higher to the number of non-zero elements. Memory allocation is a potentially expensive operation, its computational cost depends on a number of factors such as memory fragmentation and the size of the allocated chunk.

According to [32], the computational complexity of an allocator has a per-object and a per-byte component, dominated by the per-object component. A strategy to mitigate the cost of per-object allocations is using preallocation, i.e., allocating the memory needed for a complex data structure in one big allocation, in contrast to a multitude of smaller allocations. To be able do that, the number of allocations, i.e., the number of non-zeros in the matrices must be known a priori.

In our algorithm, for the radial-basis function interpolation, allocation becomes an issue when the matrices A and C are filled. For basis functions, that are defined globally, both matrices are dense and there are no non-zero entries. Otherwise, for locally defined basis functions, the number of non-zeros is not trivially known a priori. Five different methods to compute the number of non-zeros have been implemented and tested. The results are shown in Fig. 3.6.

No preallocation means to omit the explicit computation of memory allocation for the whole matrix and do the allocation whenever new elements are inserted. Albeit entries are collected for each row before committed to PETSc, resulting in much less allocations than the number of non-zeros. However, the performance is still much worse than with the other methods surveyed.

Explicit computation without using spatial indexing is the most straightforward method. The algorithm is basically traversed twice, as recommended by the PETSc manual [3]. The first time, only the number of necessary allocations and the respective matrix row are computed. On the second traversal, the actual values are set. This works well for some numerical schemes such as FEM, where the number of non-zeros can easily be computed from the type of elements and boundary conditions.

For RBF interpolation on unstructured meshes, each vertex pairing needs to be evaluated. As the filtering, by using the tree is not used here, this results in $\mathcal{O}\left(|\Xi|^2\right)$ operations for matrix C respective $\mathcal{O}\left(|\Xi| \cdot |\Delta|\right)$ operations for matrix A. This holds

true even for locally defined basis functions, where the resulting matrix is sparse. Furthermore, each operation involves a norm computation. The evaluation of the basis function itself can be omitted by computing a support radius beforehand, explicitly given or defined by a cut-off value, see Sec. 3.1.

Compute and save is an extension of the previous scheme. The distance values that are computed in the first traversal anyway are not discarded after, but saved in a matrix-like data structure. This data structure is realized as a vector with size equal to the number of rows, containing vectors with dynamically allocated entries. Both vectors are standard C++ `std::vector` data types. The second traversal then sets the previously computed values row wise. Even though this method allocates twice the amount of total memory, compared to the beforementioned method, it results in an increased performance, see Fig. 3.6.

The presumed cause is the speculative allocation usually done by implementations of C++ standard library. Given a `std::vector` of size s. An insertion of n elements, triggers the allocation of $\max(s, n)$ elements for the GCC libstd++ [5], and $\max(2s, n)$ elements for the LLVM/Clang libstd++[6]. PETSc, on the other hand, does no speculative allocations at all, which thus results in inferior filling performance compared to the C++ vector types.

Filtering using spatial indexing uses a spatial index, based on the Boost Geometry library. Analogous to the previous method, the result is saved and set in the second traversal.

Estimating the number of non-zeros is an approach that was only considered theoretically. The basic idea is to estimate a local density, based on the known bounding box and the number of vertices of the mesh. Using the support radius of the basis function, an average of covered points per basis function can be computed. In theory, this works well on isotropic, rectangular and axis-aligned meshes. For meshes that are not rectangular and axis-aligned, the bounding box does not accurately reflect the size of the mesh. For anisotropic meshes, the local density varies. Meshes that are encountered in real simulations are usually both anisotropic and unaligned, which renders this method impracticable for most use cases.

[5] `https://gcc.gnu.org/onlinedocs/gcc-8.2.0//libstdc++/api/a00599_source.html`, line 1645
[6] `http://llvm.org/viewvc/llvm-project/libcxx/trunk/include/vector?view=markup`, line 1013

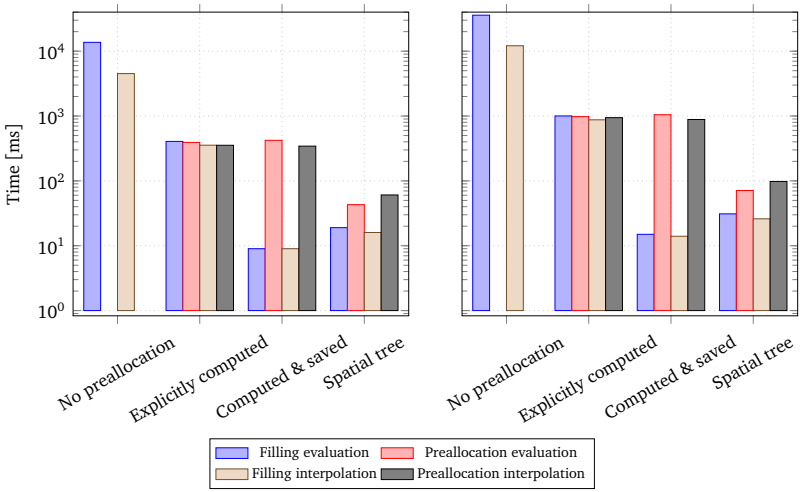


Figure 3.6: Comparison of different preallocations methods for mesh sizes 6400 and 10 000 on two ranks per participant. The plot compares times spent in the stages of preallocation and filling of matrices for evaluation (A) and interpolation (C). The total time required is the sum of all bars of one measurement. Note the logarithmic scaling of the axis.

Figure 3.6 gives an overview about the times spent in the different stages of the preallocation and filling algorithm and compares the aforementioned methods.

As a conclusion, the main challenge is posed by the non-linear nature of the problem. Unstructured grids and lacking information about local mesh density thereof, makes guessing correct values for the number of vertices that are included in the support radius next to impossible. Furthermore, there can't be any guarantees about the spatial ordering of the vertices from the solvers.

The measurements clearly show, that the effort of creating a spatial index is worth the performance gain. Initialization of the tree must be done only once at simulation startup. Furthermore, the created index can be used in other parts of preCICE, such as nearest neighbor and nearest projection interpolation methods. The Boost Geometry library helps to minimize the implementation and optimization effort of the tree index. Adapting it to preCICE internal data structures helps to keep computational and memory expensive copy operations to a minimum. The adapter was recently extended to further mesh primitives, which now can be queried for, as well.

3.4.5 Solving the Linear Systems

The need to solve a linear system arises at various points in the RBF algorithm. The weighting coefficients $\boldsymbol{\lambda}$ must be computed from $C\,\boldsymbol{\lambda} = f_\xi$. For separated or no polynomial, C either consists purely of evaluated basis functions as given in Eq. (2.28), while for integrated polynomial it has the structure as shown in Eq. (3.4).

In case of a separated polynomial, the QR-decomposition from Eq. (3.6) respectively Eq. (3.18) of the input coordinate matrix Q has to be computed. For that, the PETSc solver `KSPLSQR` is used.

To compute the rescaled interpolant, $C\,\boldsymbol{\lambda_1} = \boldsymbol{g_\xi}$ has to be solved. Beside the changed right-hand side $\boldsymbol{g_\xi}$ this is equivalent to the aforementioned solve and the solver instance can be reused, thus saving time needed to factorize C again.

The major computational effort stems from solving for the interpolation weights $\boldsymbol{\lambda}$. Direct methods, such as Gaussian-elimination or Cholesky-decomposition for positive definite matrices are not feasible due to their complexity of $\mathcal{O}\left(n^3\right)$.

PETSc offers a range of preconditioners and Krylov subspace solvers[7]. To select a suitable one, it must be implemented for the matrix type of C, which is `MATSBAIJ` for symmetric block sparse matrices and exist in a parallel version. In Sec. 4.8, different combinations of preconditioners and solvers for computing $\boldsymbol{\lambda}$ are compared.

For an efficient parallel solve, the structure of the matrices is important. Using basis functions with a compact support yields sparse matrices C and A. However, Q and V that represent the polynomial are dense. Using the integrated polynomial therefore adds global communication overhead in the process of solving the ill-conditioned system. The separated polynomial, on the other hand, solves a smaller, better conditioned system, resulting in less global communication.

As an effect of the black-box approach to coupling, we reuse the surface partitioning of the respective solver and, therefore, cannot guarantee optimal load balancing. In practice, however, the partitioning already provides a good clustering of the vertices. As a result, the matrices C and A are band-matrices and, thus, the corresponding systems are easy to solve in parallel.

[7] `https://www.mcs.anl.gov/petsc/documentation/linearsolvertable.html`

3.5 Conclusion

Six different variants to evaluate the basic RBF interpolant are presented. Either with integrated, separated or no polynomial for consistent as well as conservative interpolation. The conservative variant with separated polynomial has not been published so far, to the best of my knowledge.

The C++ implementation described in this chapter makes heavy use of two libraries: (i) PETSc is used to provide distributed data structures for matrices and vectors as well as solvers for linear systems. A re-enumeration of the vertices supplied to preCICE by the solvers is used to provide PETSc with continuously enumerated rows for each local partition. To efficiently fill the PETSc data structures, one should strive to eliminate unnecessary memory allocations. Not possessing a priori knowledge of the mesh supplied by the solver brings the need for efficient spatial queries on the mesh. Therefore, (ii) the geometry library from the Boost project, in particular its implementation of R-trees, which is designed to integrate well into existing code bases, is used. This way, it is possible to use the spatial tree on the existing mesh data structures from preCICE without expensive copy operations of vertex data. Subsequently, the tree is used for quick nearest-neighbor type queries to compute the exact number of allocations needed for each of the interpolation matrices. Once integrated, the spatial tree also proved to increase efficiency in the process to re-partitioning, which is steered by the mapping implementation. In the future, the spatial tree can furthermore become a viable building block in the process of implementing and exploring the possibilities of multiscale interpolation and other adaptive methods.

With all these functionalities and choices implemented, preCICE provides a highly optimized implementation of radial-basis interpolation in the presence of challenges imposed by the partitioned coupling approach.

4 Numerical Experiments & Results

> There are two possible outcomes: if the result confirms the hypothesis, then you've made a measurement. If the result is contrary to the hypothesis, then you've made a discovery.
>
> Enrico Fermi

In Sec. 2.3, a number of analytical results regarding the fundamental properties of interpolations with radial-basis functions are presented. These properties include convergence with decreasing mesh width, i.e., as h approaches 0, stability of the interpolation and uniqueness of the results. In spite of the analytical results, the mathematical theory of RBF interpolation is still incomplete and does not cover all important cases found in applications. For that reason, it is said, that RBF interpolation is loved by engineers, due to the ease of implementation and universal applicability and hated by mathematicians as their fundamental theory is still lacking.

In this chapter, I will use numerical methods to evaluate the interpolation behavior. Different variants of the basic algorithm will be compared. For that, appropriate test functions are used, resembling traits also found in applications. Finally, not only the interpolation itself is scrutinized, but also how common linear solver implementations cope with the resulting linear systems.

4.1 Test Setup

This section will establish a set of common parameters for the experiments. It consists of a standard domain size, test functions and methodology for error and condition number assessment. These parameters are not set in stone and for some evaluations the experiments will deviate from it when appropriate.

For the experiments, unless explicitly stated otherwise, the interpolant or input mesh is always created on the interval $[0, 1]$. The error evaluation is, however, performed on a smaller grid, e.g., $[0.08, 0.92]$ for $h_{\max} = 1/100$ and $m = 8$. This should minimize effects from

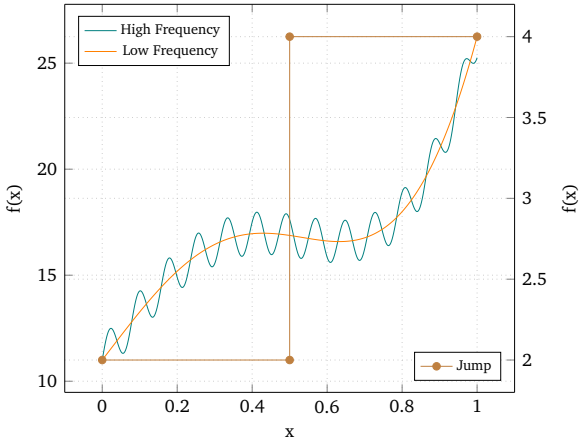


Figure 4.1: One-dimensional analytical test function variants as given in Eqs. (4.1) and (4.2)

the boundary and thus resembles the interpolation on infinite grids as closely as possible. For global basis functions with an infinite support radius, the boundary influence can not be reliably isolated from the uninfluenced inner grid. I therefore use the support radius from the local basis functions as the measure when selecting a mesh region for evaluation.

4.1.1 Numerical Test Functions

To evaluate convergence and accuracy properties, a set of analytic test functions is required. They should mimic traits that also occur in data produced by actual simulations.

Three different one-dimensional test functions $f : [0, 1] \to \mathbb{R}$ are defined:

$$\begin{aligned} f(x) &= \exp(3\,x) + 4\,\sin(5\,x) + \sin(80\,x) + 2, \\ f(x) &= \exp(3\,x) + 4\,\sin(5\,x) + 2. \end{aligned} \tag{4.1}$$

The first function combines low- and high-frequency periodic components. It also features a monotonically increasing term and a constant shift in positive y-direction. The second function, obviously, is identical up to the high-frequency term.

To analyse the influence of discontinuities, as produced by discontinues Galerkin discretization on the interpolation quality, a jump function is used in addition as a third

function:

$$f(x) = \begin{cases} 2 & x \leq 0.5, \\ 4 & x > 0.5. \end{cases} \tag{4.2}$$

To evaluate the interpolation on two-dimensional data, three different functions $[0,1] \times [0,1] \to \mathbb{R}$ have been selected.

In order to provide "realistic" data distributions, two fundamentally different standard test functions are used. Both functions have been slightly modified from the definitions found in literature so that they are defined on the unit square.

First, the so-called Eggholder function, Eq. (4.3) and Fig. 4.2 is used. It has been shifted by a constant in z-direction, as quantities from simulations such as pressure distributions normally are not zero averaged.

$$\begin{aligned} \tilde{x} &= (x - 0.5) + 512, \quad \tilde{y} = (y - 0.5) + 512 \\ f(\tilde{x}, \tilde{y}) &= -(\tilde{y} + 47) \cdot \sin\sqrt{\left|\frac{\tilde{x}}{2} + \tilde{y} + 47\right|} - \tilde{x} \cdot \sin\sqrt{|\tilde{x} - (\tilde{y} + 47)|} + 400 \end{aligned} \tag{4.3}$$

With its large number of local minima it is usually employed in testing optimization algorithms. Its highly oscillatory behavior also mimics the behavior usually found in complex flow simulations..

The Rosenbrock function, Eq. (4.4) and Fig. 4.2 is also frequently used in the field of optimization [86]. Here, it is used to compare the Eggholder function to a much smoother example.

$$\begin{aligned} a &= 1, \quad b = 100, \\ \tilde{x} &= (x - 0.5) \cdot 4, \quad \tilde{y} = (y - 0.5) \cdot 4, \\ f(\tilde{x}, \tilde{y}) &= (a - x)^2 + b \cdot \left(y - x^2\right)^2. \end{aligned} \tag{4.4}$$

As a third function, I use a two-dimensional jump function, similar to the one-dimensional equivalent in Eq. (4.2):

$$f(\boldsymbol{x}) = \begin{cases} 2 & x_1 \leq 0.5 \wedge x_2 \leq 0.5 \\ 5 & x_1 \leq 0.5 \wedge x_2 > 0.5 \\ 7 & x_1 > 0.5 \wedge x_2 > 0.5 \\ 8 & x_1 > 0.5 \wedge x_2 \leq 0.5. \end{cases} \tag{4.5}$$

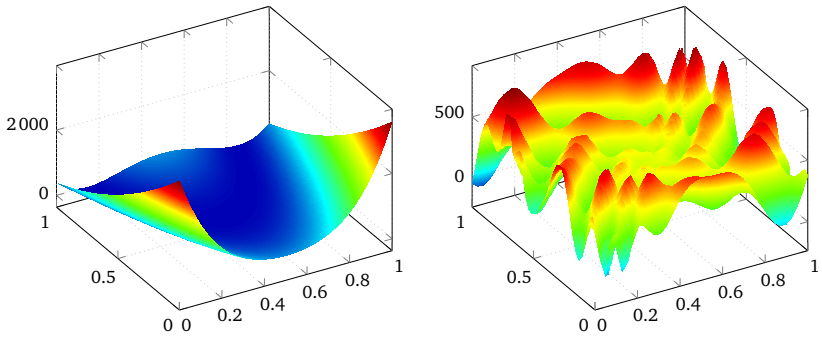


Figure 4.2: Rosenbrock (left) and Eggholder (right) three-dimensional test functions

4.1.2 Error Metrics

To assess the quality of the interpolation, a set of support points $\xi \in \Xi$ with values from the test function $f(\xi)$ is created. The interpolant s is then created using this set of support points and values. Subsequently, the interpolant is evaluated on another set of points $\delta \in \Delta$ and compared to $f(\delta)$. Some consideration when creating this test mesh must be taken into account. Due to the interpolation condition Eq. (2.26), $s(\xi) = f_\xi$, the interpolant is exact at these points. Δ should therefore not be chosen identical to Ξ, also, integral multiples of the mesh width should be avoided, as they frequently yield identical coordinates in both meshes. If the error is periodic, the density of the test mesh must be suitable to sample the error frequency f_e and thus the mesh width h_Δ should suffice $f_e < {}^1/_{h_\Delta}$ [95]. Since the error frequency is most likely non known a-priori, the test mesh should be chosen, so that $|\Delta| \gg |\Xi|$ and integral multiples should be avoided.

For the consistent interpolation, the usual metric for the pointwise error e is a suitable measure:

$$e(\delta) = s_\delta - f(\delta). \tag{4.6}$$

To acquire an integral value, the L_∞-norm gives the maximum pointwise error

$$e(\delta)_\infty = \|s_\delta - f(\delta)\|_\infty \tag{4.7}$$

or the root mean squared error (RMSE) as a weighted average

$$\text{RMSE} = \sqrt{\frac{\sum_\delta (s_\delta - f(\delta))^2}{n}}. \tag{4.8}$$

Analytical convergence proofs typically use the L_∞-norm, which I will generally use here, too, unless explicitly stated.

For conservative interpolation, this is not suitable. Foremost, the conservation needs to be fulfilled

$$\sum_\delta s_\delta = \sum_\xi f_\xi \tag{4.9}$$

which obviously conflicts with minimizing the pointwise difference of s and f, unless the two meshes have a comparable number of vertices, i.e., $|\Xi| \approx |\Delta|$.

Two different error metrics, that still produce a meaningful result for conservative interpolation, are proposed.

First, a weighted error e_w, which takes the number of elements in each mesh into account:

$$e_w(\delta) = s_\delta \cdot \frac{|\Delta|}{|\Xi|} - f(\delta). \tag{4.10}$$

This basically does a constant shift of the interpolant, based on the ratio of mesh sizes. For regular meshes, this is an appropriate offset to compensate for the constant offset that stems from the conservation condition. However, non-regular meshes may contain parts with equal (i.e., matching) mesh density, where conservation and accuracy can both be fulfilled at the same time, and parts with divergent mesh sizes, where the above-mentioned problem comes into play.

Equation (4.11) shows an alternative error formulation. It uses the rescaled interpolant s_1 which is created similar to s_δ but from the function $g(x) \equiv 1$, thus resembles $x = 1$ while showing the same conservative behavior as s_δ. The s_1 interpolant does not use a polynomial here.

$$e_r(\delta) = s_\delta \cdot \frac{1}{s_1} - f(\delta), \quad g(x) \equiv 1 \tag{4.11}$$

Figure 4.3 shows the inevitable offset of the conservative interpolant s_δ from f_ξ. The conservation condition is fulfilled by an error of only $5.7 \cdot 10^{-14}$. The rescaled interpolant, while matching the original function much better, has a conservation error of 254. The bottom plot shows the error as computed with Eq. (4.6), the weighted error e_w from Eq. (4.10) and the rescaled error e_r from Eq. (4.11). It can clearly be seen, that the local

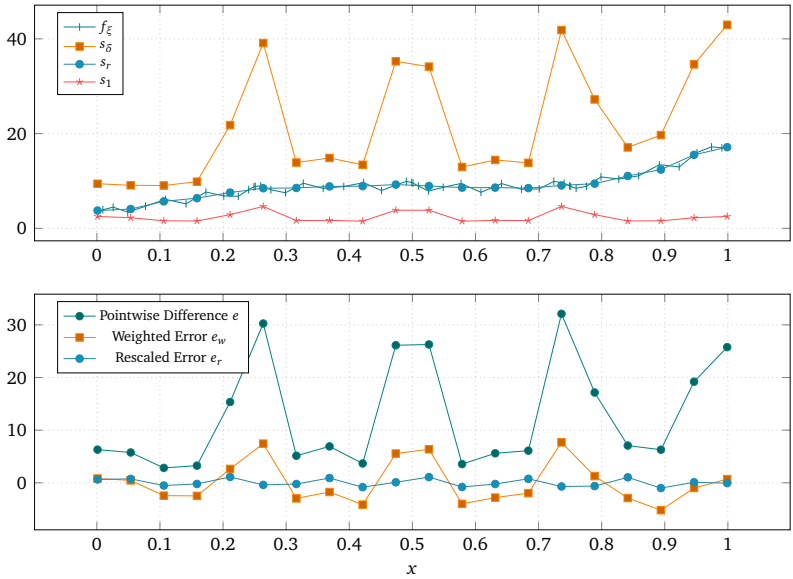


Figure 4.3: Conservative mapping from an input mesh composed of 4 Gauss-Chebyshev cells of order 12 (resulting in 48 points) to an isotropic output mesh with 20 points. The basis functions used are Gaussians with $m = 6$ together with the integrated polynomial interpolant. Top: high frequency test function f, see Eq. (4.1), evaluated at the input mesh, interpolant at the output mesh and rescaled interpolant $s_r = {}^{s}/_{s_1}$. Bottom: three different difference and error metrics.

mesh density has a great influence on pointwise difference. Due to the localized Gaussian basis function, conservation needs to be fulfilled locally. This naturally results in higher function values in places where the input mesh density is higher. The difference peaks at these locations, likewise the weighted error, merely multiplied by a constant factor.

The s_1 interpolant shows a similar behavior, depending on the mesh density as s_δ. Using it to rescale the interpolant irons out the offsets created by the conservation condition and yields a much closer, however obviously not conservative result. It can be used to create the rescaled error e_r as shown in Eq. (4.11), which gives a representation of the accuracy of the interpolant in the presence of the conservation condition.

4.1.3 Condition Number

For a numerical problem to be *well-posed* or *stable*, we require that the solution depends continuously on the input data. The condition number is a measure how the solution changes with small perturbations of the input data. These perturbation δd can originate from noisy input data or numerical errors. In the case here, I will focus on the numerical sources of errors, such as rounding errors, which are a prominent source in iterative algorithms. The condition number κ is a measure for the factor by which input errors are amplified in the solution. However, it is important to note that it does not give an exact value of the inaccuracy, but only an asymptotic upper bound.

[24, p. 406] and [112] give the rule of thumb, if the condition number is $\kappa = 10^k$, then one may lose up to k digits of accuracy when solving the linear system.

According to [82, p. 34], given a perturbed problem

$$F(x + \delta x, d + \delta d) = 0 \tag{4.12}$$

of the original problem $F(x, d) = 0$ with input data d and solution x, we require that

$$\begin{aligned} &\exists \eta_0 = \eta_0(d) > 0, \exists K_0 = K_0(d) \text{ such that} \\ &\text{if } \|\delta d\| \leq \eta_0 \text{ then } \|\delta x\| \leq K_0 \|\delta d\| \end{aligned} \tag{4.13}$$

which means we require the perturbation of the solution $\|\delta x\|$ to be bound by $K_0 \|\delta d\|$. From this, an *absolute* and *relative condition number* can be derived. The absolute condition number is given by

$$\kappa_{\text{abs}}(d) = \sup \left\{ \frac{\|\delta x\|}{\|\delta d\|}, \delta d \neq 0, d + \delta d \in D \right\} \tag{4.14}$$

for the set D of viable input parameters.

Here, I am looking at the case of solving the interpolation matrix C for the weights $\boldsymbol{\lambda}$ of the RBF interpolation, given as the linear system in Eq. (2.27):

$$C \cdot \boldsymbol{\lambda} = \boldsymbol{f}_\xi$$

with the right-hand side $\boldsymbol{f}_\xi$ representing the potentially distorted input data. The condition number for linear systems, with respect to a perturbed right-hand side can be defined as

$$\kappa(\boldsymbol{f}_\xi) \simeq \frac{\left\|C^{-1}\right\| \left\|\boldsymbol{f}_\xi\right\|}{\left\|C^{-1}\boldsymbol{f}_\xi\right\|} \leq \|C\| \left\|C^{-1}\right\| = \kappa(C). \tag{4.15}$$

In the upcoming numerical experiments, the Python function `numpy.linalg.cond(x, p=None)` is used to compute the condition number.

The algorithm is based on the fact that, if the Euclidean norm L_2 is used in the equations above (which means p=None or p=2 as an argument to `numpy.linalg.cond`), the condition number can be expressed in terms of the singular values σ

$$\kappa(C) = \frac{\sigma_{\max}(C)}{\sigma_{\min}(C)}, \tag{4.16}$$

where $\sigma_{\max}(C)$ and $\sigma_{\min}(C)$ are the largest and smallest absolute value, respectively, of all singular values of C.

This direct computation of the condition number hence requires a computational costly singular value decomposition, which is still feasible for small experiments.

4.1.4 Implementation

The numerical experiments are performed using PyRBF[1], a Python implementation of RBF interpolation using the well-known libraries NumPy [107], SciPy [64] and Pandas [75]. The linear systems are solved using `numpy.linalg.solve` which in turn calls the LAPACK routine `_gesv`. LAPACK [2] employs a LU-decomposition with partial pivoting to solve the linear system.

PyRBF implements the same algorithms that are found in preCICE and additional enhancements for experimentation, that may eventually end up in preCICE as well. It focuses on extensibility and flexibility, rather than performance.

[1] `https://github.com/floli/PyRBF/`

4.2 Standard RBF Interpolation

In this section, I will examine the behavior of the standard RBF interpolant Eq. (2.25). This formulation is devoid of extensions, such as additional polynomials, rescaling or an alternative basis. It will serve to form a baseline to which more ambitious implementations can be compared. In the following, I will assess accuracy and condition number in the case of decreasing mesh width h, examine how discontinuities and different frequencies of the input data influence the interpolating and observe the influence of the basis function's different shape parameters. The effect of error saturation will be analyzed and how the condition number varies with mesh sizes and shape parameters. Finally, the interpolation on non-uniform and two-dimensional meshes will be assessed.

Mesh Convergence for Different Basis Functions Figures 4.4 to 4.6 show the L_∞-error as the mesh width h approaches 0 for various combinations of basis functions and test functions.

The interpolation using the multiquadrics basis function was not able to converge to a stable solution for a range of shape parameters $\epsilon = 0.1 \ldots 16$, Fig. 4.4 only show the largest and smallest of the shape parameters assessed. The instability is indicated by the chaotic fluctuations in the solution and the high condition number of $\kappa \approx 10^{20}$.

The accuracy of thin-plate splines and volume splines basis functions significantly depend on the test function used. It is best for the constant function and gets worse from low-frequency over high-frequency to the jump function.

If we look at the analytical investigations as summarized in Sec. 2.3.2, we see, that with the given setup, none of the basis functions can be considered of thin-plate spline type, following the definition from Eq. (2.38). The absence of the polynomial in one-dimensional space yields $k = 0$ and $d = 1$. The convergence results from Eq. (2.39) then only apply to basis functions of the form r^{-1}, which is not the case here. In addition, it is unclear, whether the underlying mathematical theory holds for cases without a polynomial, as some basis functions need it to be uniquely solvable. I will revisit this fact in Sec. 4.3 when looking at the results of interpolation with polynomials. Still, the mentioned convergence rates of $h\sqrt{\log(h^{-1})}$ and $\sqrt{h}$, for $2k - d = 2$ or $2k - d = 1$, respectively or h otherwise, can be considered an upper bound up to an unknown constant factor for the case under investigation. For some test functions, however, the actual error is much lower. The first example is the volume spline basis function interpolating the constant test function, where the L_∞-error remains in the range of 10^{-15} and at this level is probably influenced by the machine precision. The second example is the discontinuous jump test function. All basis

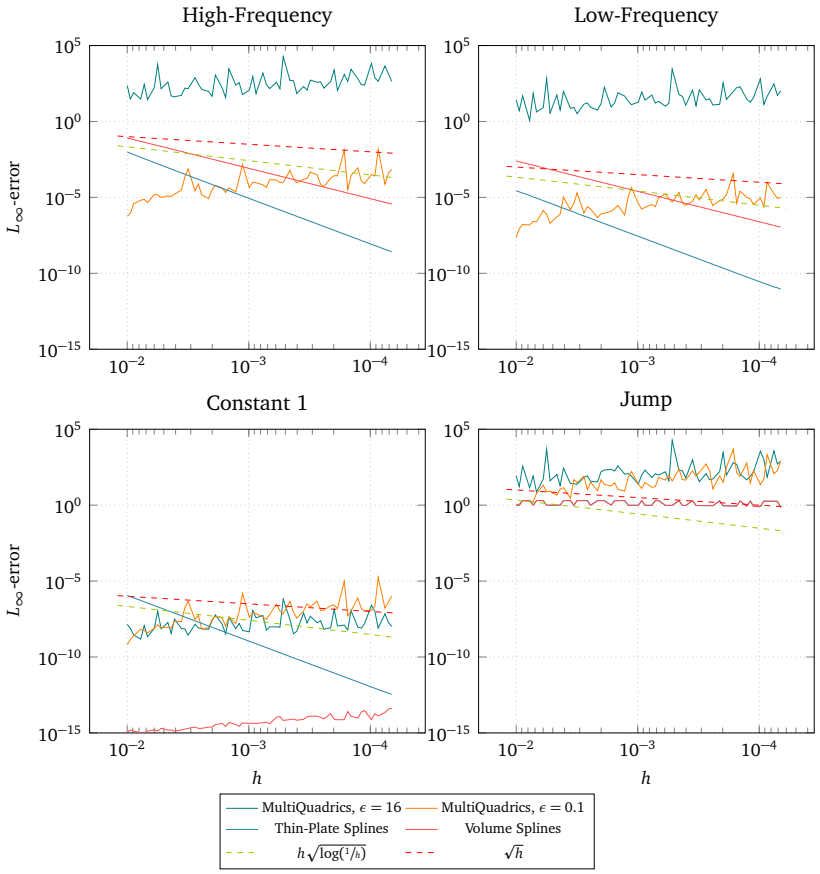


Figure 4.4: RBF interpolation for different test functions and basis functions. The L_∞-error for increasing mesh resolution and multiquadrics, thin-plate splines and volume splines basis functions is shown. Test functions used are the high- and low-frequency variants, Eq. (4.1) and the discontinuous jump function Eq. (4.2). Additionally, a constant function $g(x) \equiv 1$ is used as a test function. Dashed lines show the theoretical error in unbounded domains (up to a unknown constant).

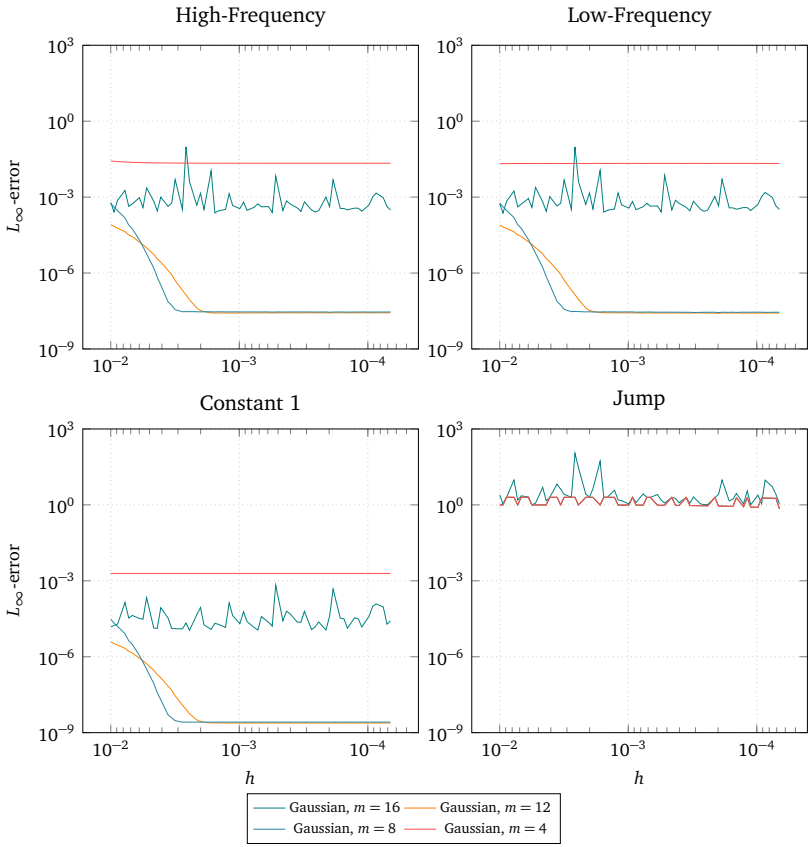


Figure 4.5: RBF interpolation for different test functions and Gaussian basis functions. The L_∞-error for increasing mesh resolution and different shape parameters m is shown. Test functions used are the high- and low-frequency variants, Eq. (4.1) and the discontinuous jump function Eq. (4.2). Additionally, a constant function $g(x) \equiv 1$ is used as a test function.

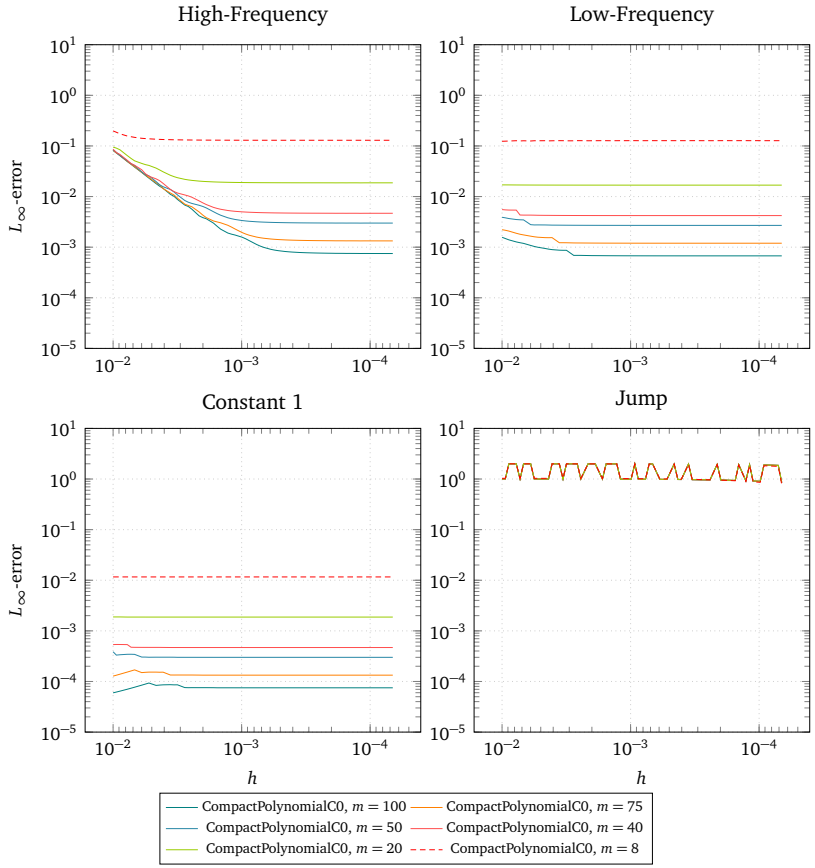


Figure 4.6: RBF interpolation for different test functions and Compact Polynomial C0 basis functions. The L_∞-error for increasing mesh resolution and different shape parameters m is shown. Test functions used are the high- and low-frequency variants, Eq. (4.1) and the discontinuous jump function Eq. (4.2). Additionally, a constant function $g(x) \equiv 1$ is used as a test function.

functions perform exceptionally bad when confronted with discontinuities. The almost regular up and down pattern of the volume and thin-plate spline basis functions can be attributed to interference between the input mesh Ξ and the test function, as some specific, recurring offset between the discontinuity and the mesh vertices happens to slightly improve the interpolation, compared to other offsets.

Figures 4.5 and 4.6 show the error for the Gaussian and Compact Polynomial C0 type basis function for a selection of support radii. Both show a similar behavior, fundamentally different from the global basis functions. Gaussians show usefulness only in a narrow range of shape parameters m. A value of $m = 4$ results in sharp basis function defined only one a small support radius. This results in an inadequate density of basis functions and inferior error rates, even for the constant test function. On the other hand, a support radius which is too large results in unstable behavior of the linear solver, as indicated by random fluctuations and an average condition number of $\kappa = 5 \cdot 10^{12}$. The case of too small support radii can be improved by using the rescaling technique, see Sec. 4.4.

Though graphically, their shape is very much equal to Gaussians, compact polynomial type basis functions exhibit a vastly different behavior regarding stability and accuracy. While compact polynomials are stable up to much higher values of m, compared to Gaussians, the achieved accuracy is significantly lower, even with a much larger support radius. The influence of the different test functions is more pronounced then with Gaussians, but less than with the global basis functions.

Interpolation of Discontinuous Functions Figure 4.7 illustrates the effects of boundaries and discontinuities on the interpolant and the coefficients of $\boldsymbol{\lambda}$. For localized basis functions, a distance of the support radius is therefore sufficient to exclude boundary effects from the interpolation.

For global basis functions, such as thin-plate splines and volume splines, the situation is more diverse. In contrast to their global nature, the effect of boundaries and discontinuities is still very localized. Discontinuities, however, can lead to large coefficients, both negative and positive, near the locus of the discontinuity.

Good interpolation of functions that include discontinuities pose a challenge for RBF interpolation, regardless of the basis used. The L_∞-error on the entire mesh (including boundary) compared to a mesh that excludes the discontinuity, i.e., $[0, 1] \backslash [0.4, 0.6]$ is up to several orders of magnitude larger. The numbers range from a factor of ≈ 7 for Gaussians, ≈ 10 for compact polynomials, ≈ 91 for thin-plate splines to $\approx 4 \cdot 10^{14}$ for volume splines. This extremely wide range of ratios is not caused by superior handling of the discontinuity

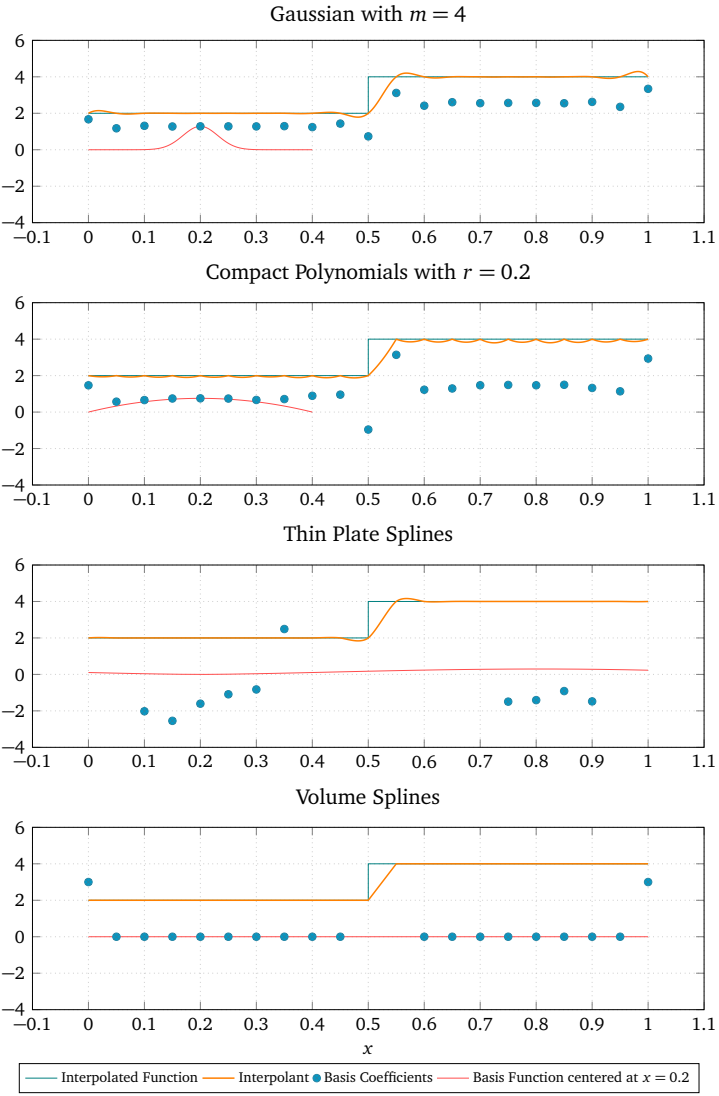


Figure 4.7: RBF interpolation for the jump function to analyze the effects from boundaries and discontinuities on the interpolants and weights of the basis functions $\boldsymbol{\lambda}$ (brown dots). Additionally, the scaled basis function at $x = 0.2$ is shown. Note that some weights are omitted, due to the y-axis scaling.

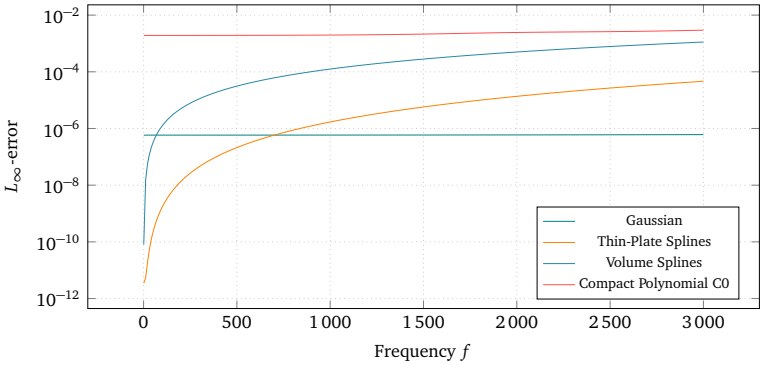


Figure 4.8: RBF interpolation of a test function $g(x) = \sin(f \cdot x)$ with increasing frequency f. The error is shown for Gaussian with $m = 6$, for Compact Polynomial C0 with $m = 40$, for Volume Splines and Thin-Plate Splines. To capture the high-frequency oscillations a fine input mesh on $[0, 1]$ consisting of 10 000 nodes and a test mesh on $[0.3, 0.7]$ of 23 000 mesh nodes are used.

of, e.g., volume splines, but is the consequence of better interpolation of the constant part of the function. If we look at the errors only in the neighborship of the discontinuity, i.e., $[0.45, 0.55]$, all basis functions yield surprisingly equal and high numbers of $L_\infty \approx 1.99$.

Frequency Dependency Observations made in the previous section point at a frequency dependency on the test function of the interpolation. Figure 4.8 shows how the L_∞-error varies with the frequency of the test function. For that, the simple function $g(x) = \sin(f \cdot x)$ was interpolated on $[0, 1]$ and evaluated on $[0.3, 0.7]$ to ignore boundary effects.

The global basis functions, i.e., volume splines and thin-plate splines exhibit a strong dependency on the frequency of the test function. The error increases monotonically with the frequency showing an asymptotic behavior. Compact polynomials are only minimally influenced, converging to a similar L_∞-error as the volume splines. The Gaussian basis functions, on the other hand, show not to be influenced by the frequency at all.

Error Saturation & Shape Parameter for Compact Polynomials & Gaussians Figures 4.5 and 4.6 furthermore show the effect of error saturation, a stagnating error, even as h decreases. Theoretical considerations of the effect are detailed in Sec. 2.3.3. The effect is not only visible with Gaussian basis functions, but also for the compact polynomials. The test function used, with the exception of the discontinuous jump, show only little influence

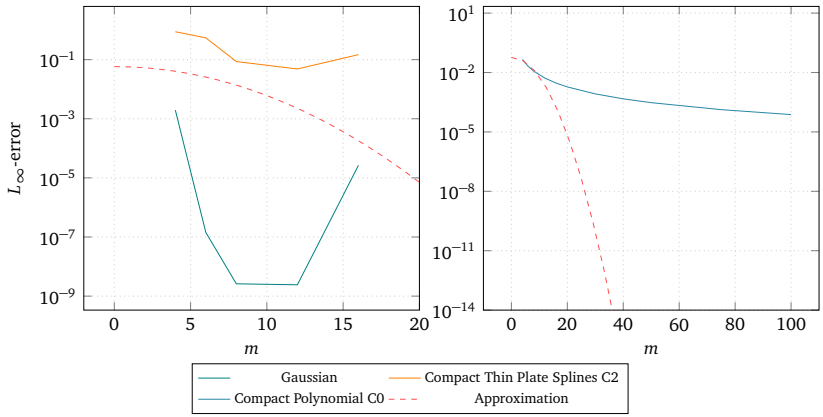


Figure 4.9: Influence of the shape parameter m on the L_∞-error of the interpolation. Numerical results for Gaussians, Compact Thin-Plate Splines and Compact Polynomial C0 are shown together with the approximation according to Eq. (2.49) (dashed lines). Mesh width is $h = 6.6 \cdot 10^{-5}$ (15 000 mesh nodes on $[0, 1]$).

on error saturation. A Gaussian basis function with a small support of $m = 4$ is unable to adequately resolve the test functions and achieved L_∞-errors of 0.002 to 0.02. Choosing a large support of $m = 16$, on the other hand, results in an unstable interpolation. For interpolations with $m = 12$ and $m = 8$, an error saturation of $2.9 \cdot 10^{-8}$ to $2.4 \cdot 10^{-9}$ can be observed. In contrast to the results in Sec. 2.3.3, the final accuracy is almost identical, not depending on the support radius.

Figure 4.9 shows how the accuracy changes with the support radius for compact basis functions. As already mentioned above, Gaussians and also compact thin-plate splines achieved a stable interpolation only in a small range of support radii.

Compact polynomials remain stable up to a large support of $m = 100$, though the gain in accuracy diminishes.

In all cases, the approximation from Eq. (2.49) does not provide a sufficiently accurate estimation of the error saturation as a function of the support radius.

Condition Number for Different Basis, Shape Parameters & Mesh Resolutions
Figure 4.10 shows the condition number κ for multiquadrics, volume splines and thin-plate splines global basis functions as well as for Gaussians and compact polynomials local basis functions with different values of m. Since the condition number is independent of the test

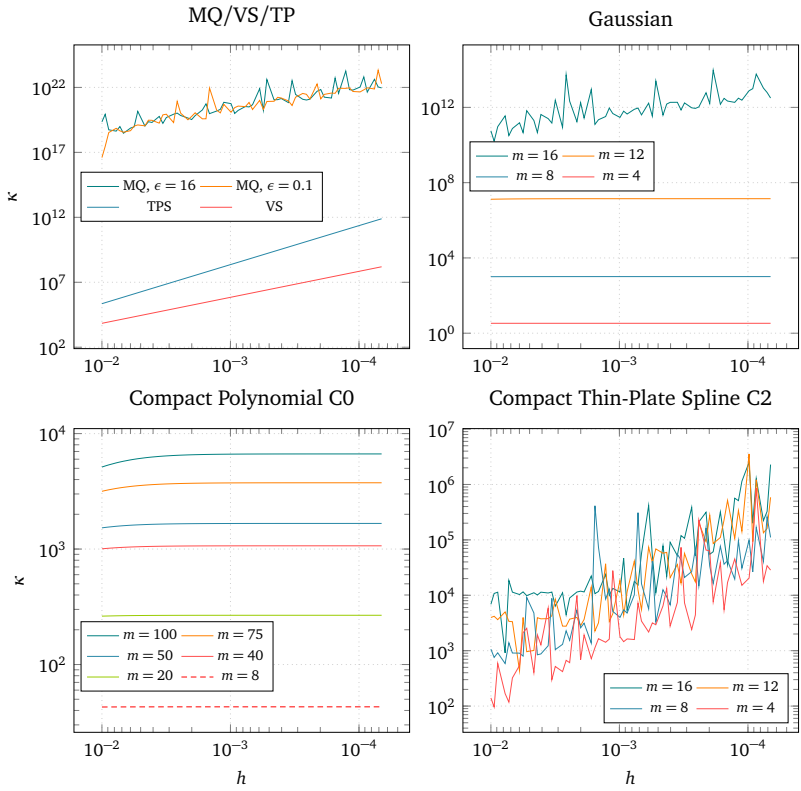


Figure 4.10: Condition number κ of the interpolation matrix C for different basis functions and shape parameters over the mesh width h. Consistent interpolation with standard radial-basis functions is used.

function, i.e. the right-hand side of the respective linear system, it is only shown for one test function.

For the two local basis functions, Gaussians and compact polynomials, the condition number remains almost constant over h. As I use m to control for the shape parameter, the basis functions width remains constant relative to the mesh. The support radius, expressed in the number of vertices proves to be an adequate measure to control the condition number.

For all local basis functions κ increases with the support radius. This behavior limits measures to improve the accuracy by increasing the support radius, as the interpolation eventually becomes unstable. This can be seen here for the multiquadrics basis functions.

For the compact polynomials the condition number remains low, compared to Gaussians. This is the case even for large support radii. However, as we have previously seen, the error remains high.

Global basis functions show a moderate increase of the condition number with the mesh size. Likewise with local basis functions, this can be seen as an increase of vertices included in their support radius and thus showing a consistent behavior of increasing κ with m.

Influence of Non-Uniform Meshes The preceeding measurements revealed that the support radius relative to the mesh density plays a tremendous role for accuracy and stability of the interpolation. Especially Gaussian basis functions are susceptible. For uniform meshes, i.e., where $h = h_{\max}$ holds, finding an appropriate value for m is easy. For non-uniform meshes, i.e., with non-constant h, this is a daunting task. Non-uniform meshes are, however, used regularly in practice, e.g., in meshes refined at certain feature edges of the geometry or solvers based on the discontinuous-Galerkin method. These type of solvers use a Gauss-Chebyshev distribution of nodes. The position of the k-th, $k \leq p$ vertex inside a cell is given by

$$x_k = \cos\left(\frac{2k-1}{2p}\pi\right), \quad k = 1, \ldots, p \tag{4.17}$$

for polynomial degree p in the one-dimensional interval $(-1, 1)$.

Figure 4.11 illustrates the non-uniform mesh width of two cells with Gauss-Chebyshev vertex distribution of polynomial degree $n = 24$ and how the the local support radius, expressed in terms of m varies. Furthermore, it shows how the pointwise error of an interpolation using Gaussian basis functions (with a constant shape parameter based on $m = 4$ at $h_{\max}$) depends on the local mesh density.

The number of vertices inside one cell is equivalent to the polynomial degree p used.

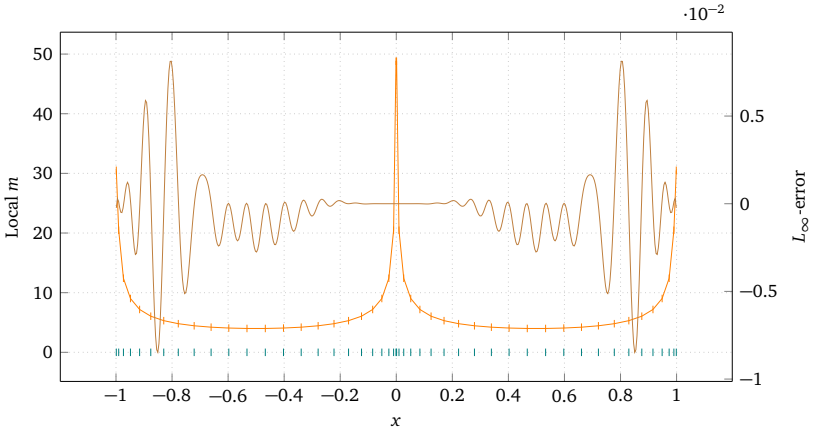


Figure 4.11: Illustration of two cells with a Gauss-Chebyshev vertex distribution for polynomial order $p = 24$ (blue). In red, the local support radius in terms of m for a constant shape parameter ϵ is shown, with $m = 4$ for $h_{\max} \approx 0.065$ at $x = \pm 0.5$. Additionally, brown (left y-axis) shows the pointwise error of an interpolation using Gaussian basis functions.

To define the shape parameter, the maximum mesh width $h_{\max}$ is used. For a given basis function and shape parameter ϵ, a local m can be defined, with $m = m_{\min}$ at $h_{\max}$ and $m_{\max}$ at $h_{\min}$. The relation of the maximum to the minimum mesh width is not constant as p goes up, but approximately $h_{\max}/h_{\min} \approx p/2$.

Figure 4.12 shows the behavior of an interpolation using Gaussian basis functions as the polynomial order of the underlying Gauss-Chebyshev vertex distribution increases from 4 up to 56. For $m = 4$ and $m = 6$, two regions can clearly be distinguished. Up to approximately $m_{\max} \approx 100$ convergence can be observed, regardless of the polynomial degree. After that, the interpolation becomes unstable. For $m = 8$ the separation is not as clearly visible, but still existing. Naturally, the interpolation becomes unstable at higher polynomial degrees determining the vertex distribution, as $m_{\max}$ increases, even if $m = m_{\min}$ is held constant. In Fig. 4.5, it can be seen that the interpolation on uniform meshes becomes unstable between values for m between 12 and 16. In contrast to this, the number of vertices contained in the support radius, can be much higher for non-uniform meshes, as long as the increased mesh density is limited to only a part of the input mesh.

Compact polynomial basis functions behave analogously and consistent to the results presented previously in this section, as can be seen in Fig. 4.13 The interpolation is stable

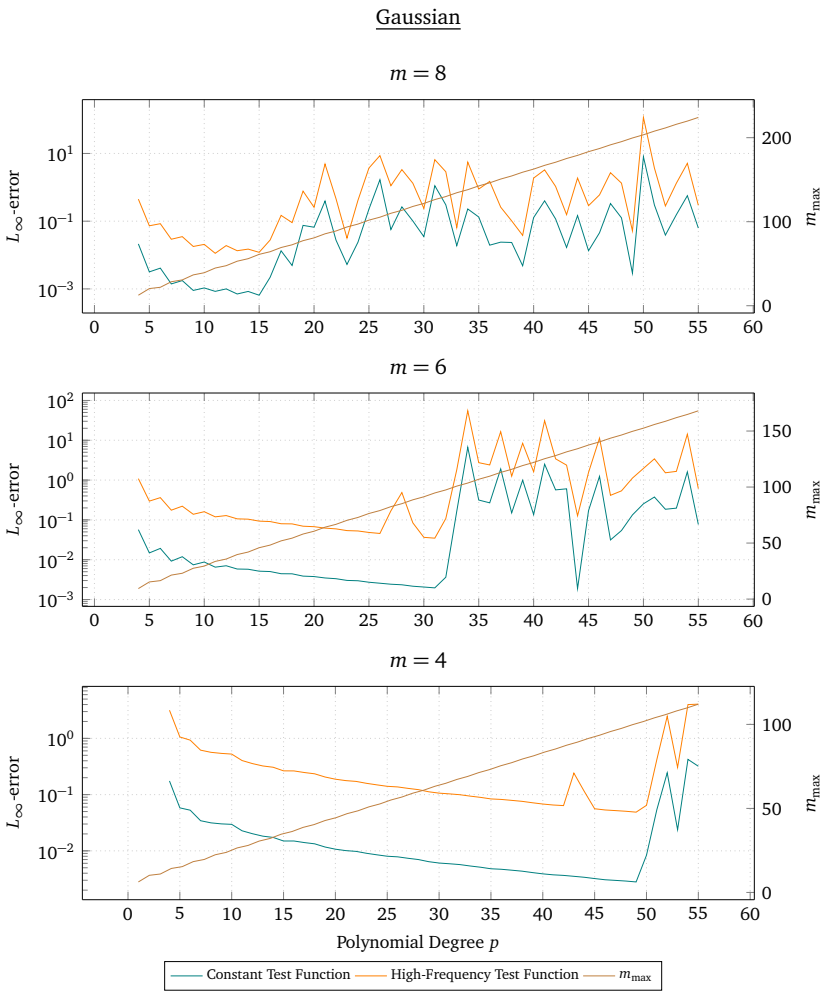


Figure 4.12: RBF interpolation on a mesh with Gauss-Chebyshev distributed vertices of two test functions using a Gaussian basis for different values of m. The L_∞-error and $m_{\max}$, computed from $h_{\min}$ and the given m are shown. The mesh cell size is 0.0625 on $[0, 1]$, resulting in 16 cells with polynomial of degree p.

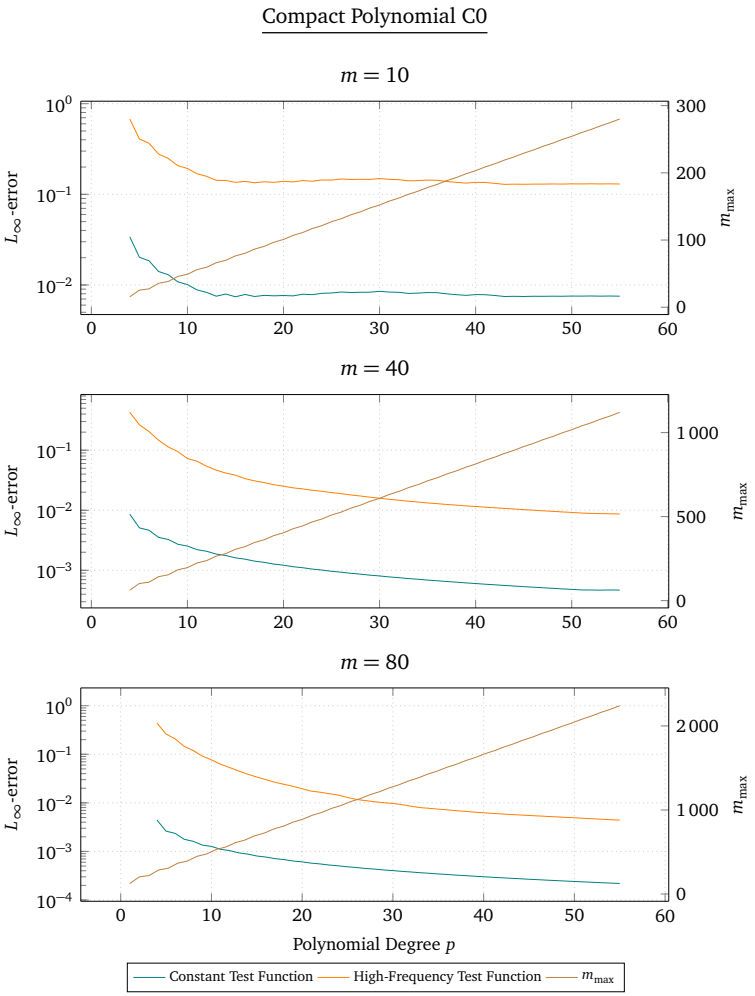


Figure 4.13: RBF interpolation on a mesh with Gauss-Chebyshev distributed vertices of two test functions using a Compact Polynomial C0 basis for different values of m. The L_∞-error and $m_{\max}$, computed from $h_{\min}$ and the given m are shown. The mesh cell size is 0.0625 on $[0, 1]$, resulting in 16 cells with polynomial of degree p.

up to much larger support radii. In fact, no unstable interpolation occurred up to values of $m = m_{\text{min}} = 80$ and $m_{\text{max}} = 10\,400$ for a polynomial degree determining the vertex distribution of $p = 256$. While the accuracy was consistently better than with Gaussians in the stable regime, it remained low with the L_∞-error experienced saturation at about 10^{-4} for the constant test function and $5 \cdot 10^{-3}$ for the high frequency test function. For the given mesh setup, the evaluation of larger values of m is not advisable, as the mesh sizes for 16 cells of polynomial degree $p = 256$ boundary effects become dominant for large values of m.

Two-dimensional Interpolation Interpolation with radial-basis functions is dimension agnostic in theory. By evaluating the basis function on the norm of the distance of two points, the dimensionality of the input or output mesh is effectively taken out of the algorithm.

In this section, I will give an overview on results for two-dimensional meshes and how far the results achieved so far for one-dimensional interpolation apply to two-dimensional cases.

In Fig. 4.14, the result and the point-wise error from the interpolation of two-dimensional test functions is shown. For the definition and visualization of the test functions used, see Eqs. (4.3) to (4.5) and Fig. 4.2.

The result of the piecewise-constant (jump) test function shows similar traits compared to one-dimensional interpolation. The largest errors occur at the locus of the discontinuities. Elsewhere, the error is constant on a moderate level.

For the Rosenbrock function, a low error level was achieved for the inner, lower-gradient parts of the test function. Going towards the border of the interpolation regime, two effects become visible. First, the usual decay in accuracy towards the boundary, even for lower-gradient regions, such as around $x = 0.5$ and $y = 1$, is seen as a ridge parallel to the boundary. Second, the high-gradient corners of the Rosenbrock function cause significant negative peaks in the error, i.e., the interpolation underestimates the real function value. The shape of the Rosenbrock function is particularly problematic here, as the high-gradient regions occur near the boundaries, were the effect of insufficient coverage by the basis functions also comes into play.

The highly oscillatory nature of the Eggholder function obviously posed a challenge to the interpolation, as can be seen by the highest error among the three test functions. In the error plot, diagonal lines the error stick out. It is currently not known, what causes this effect. Their existence is independent from different resolutions of the input and output meshes, different support radii or even different basis functions.

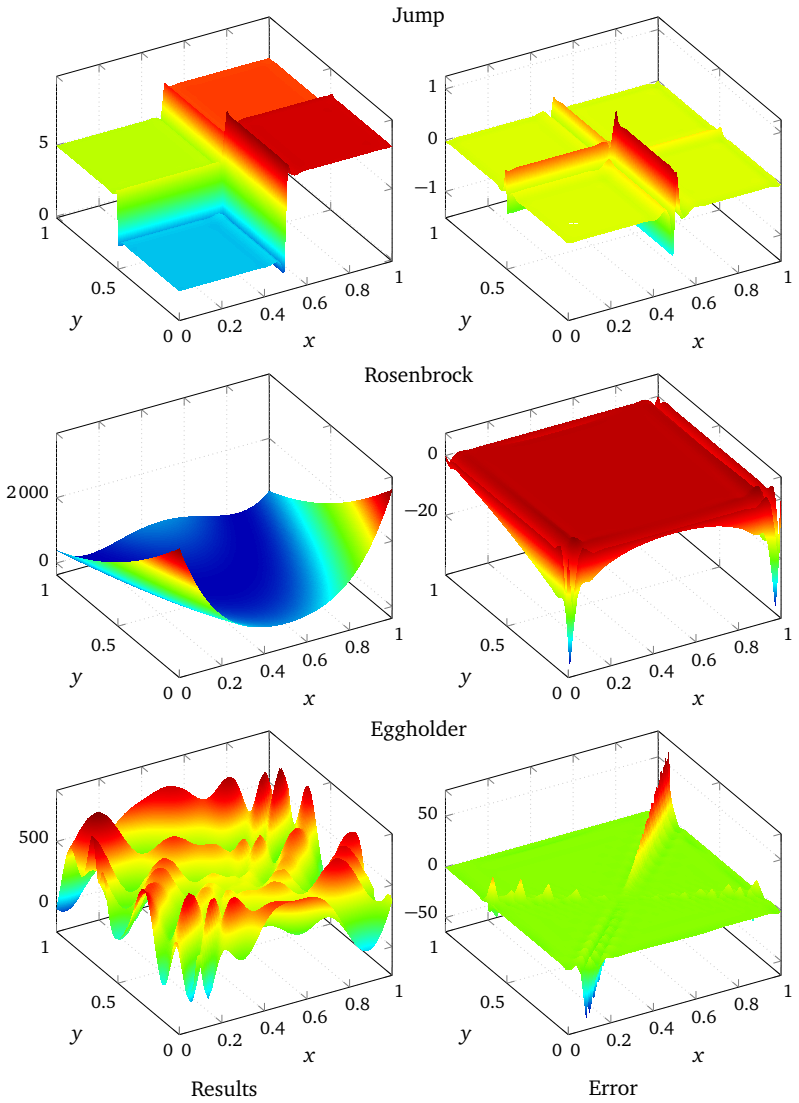


Figure 4.14: Two-dimensional RBF interpolation for a mesh with 100^2 to 60^2 elements for the three test functions Eqs. (4.3) to (4.5). Gaussian basis functions with $m = 8$ are used. The result (left) as well as the point-wise error (right) is shown.

Test Function	m	L_∞-error	RMSE	m_{opt}	L_∞-error at m_{opt}	RMSE at m_{opt}
Jump	8	1.2835	0.0384	5	1.1691	0.0337
Rosenbrock	8	35.9896	0.7585	16	0.1951	0.0226
Eggholder	8	63.8469	1.2790	8	63.8469	1.2790

Table 4.1: Interpolation results for two-dimensional basis functions. The optimal value for m is based on the minimum RMSE.

Table 4.1 lists the results for $m = 8$ and for an optimal value of m that achieved the lowest RMSE for a test function. The jump-function works best with a small support radius of 5 or even 4 vertices (which achieved a slightly better L_∞-error). A small support radius helps to better resolve the unsteady jump. The Rosenbrock function, on the other hand, achieved best results using a large support radius of 16 vertices. An even larger support radius of $m = 18$ achieved a slightly better result for the L_∞-error.

4.3 Integrated and Separated Polynomial

In the previous section, the RBF interpolation was implemented without an additional polynomial, in its simplest form as given in Eq. (2.25). In Sec. 2.3.1, the standard interpolant is extended by an additional polynomial. The polynomial can either be integrated in the interpolation matrix or evaluated separately, as described in Sec. 3.2. Unless stated otherwise, the polynomial is of first degree, i.e., linear here.

Numerical results confirm that the two implementation variants of the polynomial, integrated (see Eq. (3.4) and Eq. (3.9)) and separated (see Eq. (3.8) and Eq. (3.11)), while mathematically not equivalent, yield almost identical results. For that reason, I will resort to evaluate only the separated variant here, unless explicitly stated.

Mesh Convergence for Different Basis Functions Looking again at the analytical results for convergence behavior in Sec. 2.3.2, we find that for the one-dimensional case with a linear polynomial ($d = 1$ and $k = 2$), the thin-plate spline type functions according to Eq. (2.38) are of the form $\varphi = r^{2k-d} = r^3$. For these type of functions, a convergence rate of $\|s - f\|_\infty \leq c \cdot h$, according to Eq. (2.39) can be expected. The actual basis functions used, e.g., $\varphi_{\text{TPS}} = \|x\|^2 \log(\|x\|)$, do not match the definition of Eq. (2.39). The mentioned convergence rates can therefore only give an orientation, but the mathematical proof, however, is not applicable here.

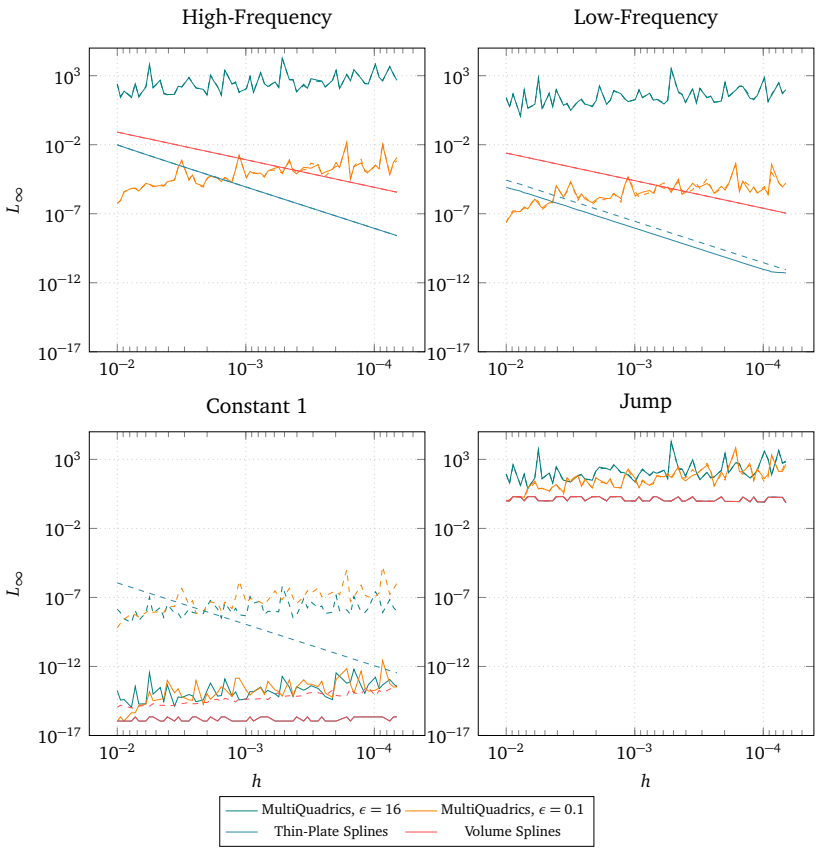


Figure 4.15: RBF interpolation using a separated polynomial for different test functions and basis functions. The L_∞-error for increasing mesh resolution and multiquadrics, thin-plate splines and volume splines basis functions is shown. Test functions used are the high- and low-frequency variants, Eq. (4.1) and the discontinuous jump function Eq. (4.2). Additionally, a constant function $g(x) \equiv 1$ is used as a test function. Dashed lines show the error of the respective interpolation without a polynomial for comparison.

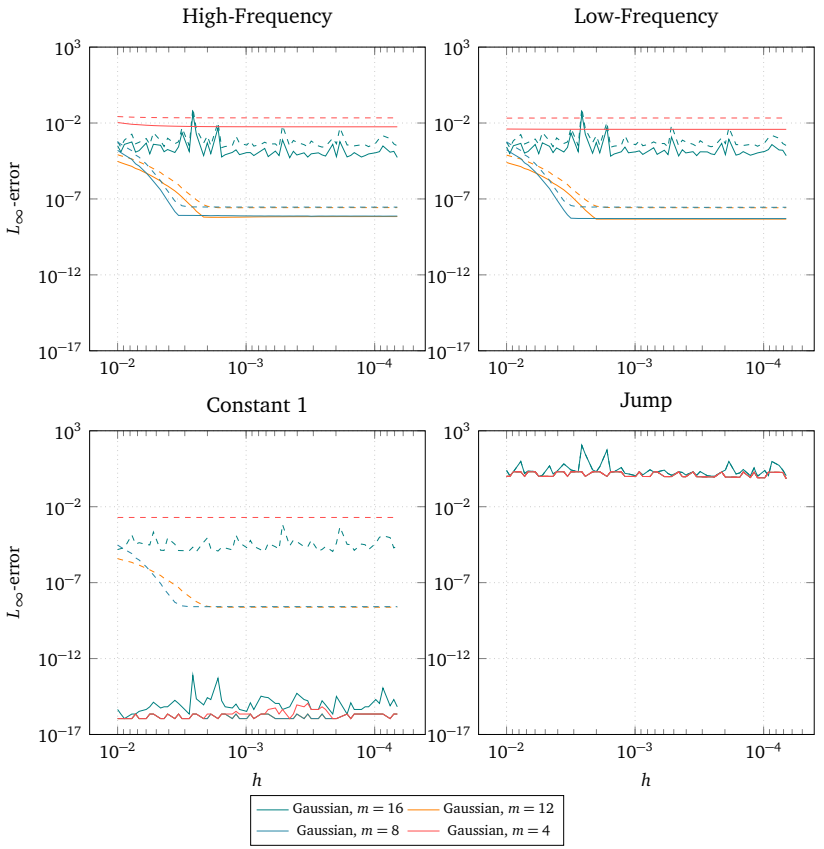


Figure 4.16: RBF interpolation with separated polynomial for different test functions and Gaussian basis functions. The L_∞-error for increasing mesh resolution and different shape parameters m is shown. Test functions used are the high- and low-frequency variants, Eq. (4.1) and the discontinuous jump function Eq. (4.2). Additionally, a constant function $g(x) \equiv 1$ is used as a test function. Dashed lines show the error of the respective interpolation without a polynomial for comparison.

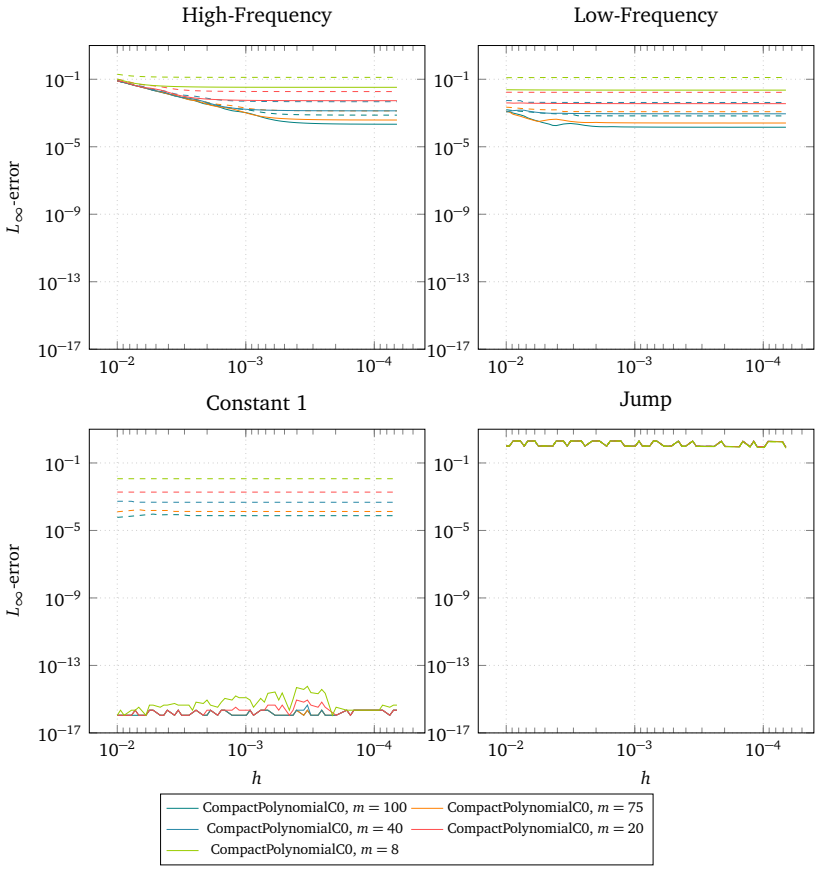


Figure 4.17: RBF interpolation with separated polynomial for different test functions and Compact Polynomial C0 basis functions. The L_∞-error for increasing mesh resolution and different shape parameters m is shown. Test functions used are the high- and low-frequency variants, Eq. (4.1) and the discontinuous jump function Eq. (4.2). Additionally, a constant function $g(x) \equiv 1$ is used as a test function. Dashed lines show the error of the respective interpolation without a polynomial for comparison.

Figs. 4.15 to 4.17 show the L_∞-error as the mesh width h decreases for RBF interpolation using a separated polynomial.

The most notable difference between the standard interpolant and the interpolant with the polynomial added can be seen with the interpolation of the constant test function. All basis functions achieve an extremely low error error of 10^{-16}, with the unstable multiquadrics and Gaussian with $m = 16$ oscillating around 10^{-13}. This comes as little surprise, the polynomial can perfectly approximate the constant function. The computed weights $\boldsymbol{\lambda}$ for the basis functions are all close to zero then. The same, obviously, is true for any linear polynomial test function. Apart from the constant test function, the additional polynomial has only little influence on the results.

A possible explanation for the negligible effect is, that all test functions are relatively close to zero at the test interval. This, however, is only partially confirmed, as seen in Fig. 4.18. There, the implementations with and without polynomial are compared with respect to the high frequency test function undergoing a constant shift in y-direction. Gaussians and compact polynomial basis function are directly and in the same way influenced by the shift Δy as the error grows with the shift when the RBF interpolant is not augmented by an polynomial. With a polynomial, however, the effect of the shift in y-direction is completely neutralized, as the error is constant, independent from the shift. The two global basis functions, volume and thin-plate splines show a different behavior. Volume splines, regardless of whether a polynomial is added or not, are unaffected by the shift. This can also be seen in the close to zero values in the lower plot of Fig. 4.18. Thin-plate splines, on the other hand, show a behavior similar to the compact basis functions, as the error increases with the shift, albeit on a much smaller level. Most notably, unlike for the other basis functions, the polynomial has an slightly adverse effect on the error when using thin-plate splines. This is not immediately visible from the plot, since only the absolute value of the error difference is plotted, so that a logarithmic scale can be used. At moderate shifts in the range of $\Delta y \approx -120 \dots 60$, the error deteriorates by using the polynomial. Outside that range (not shown in the plot), the effect is inversed and the polynomial improves the results.

Error Saturation & Shape Parameter for Compact Polynomials & Gaussians
The effect of error saturation when the the compact basis function are used is also present with an RBF interpolant augmented by a polynomial. The absolute value of the saturation error slightly improves when a polynomial is used by less than one order of magnitude, compare Figs. 4.16 and 4.17. Beyond that, the polynomial shows only little influence on

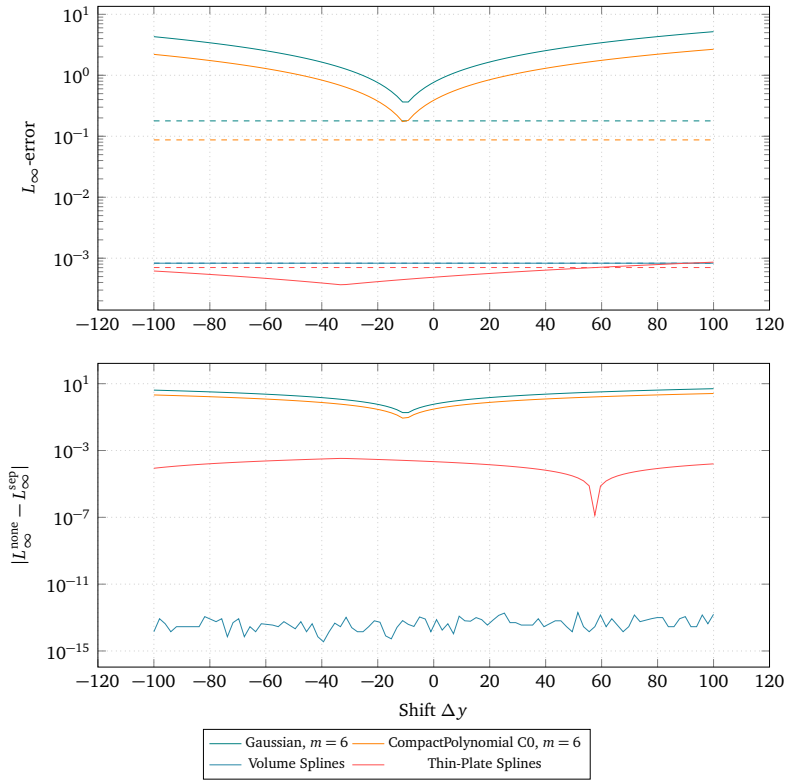


Figure 4.18: Comparison of interpolation with and without polynomial under a constant shift Δy of the high frequency test function. At the top plot, the dashed lines indicate the interpolation using the separated polynomial, while the solid lines are without any polynomial. The bottom plot shows the absolute difference between the errors of interpolation with and without polynomial over Δy.

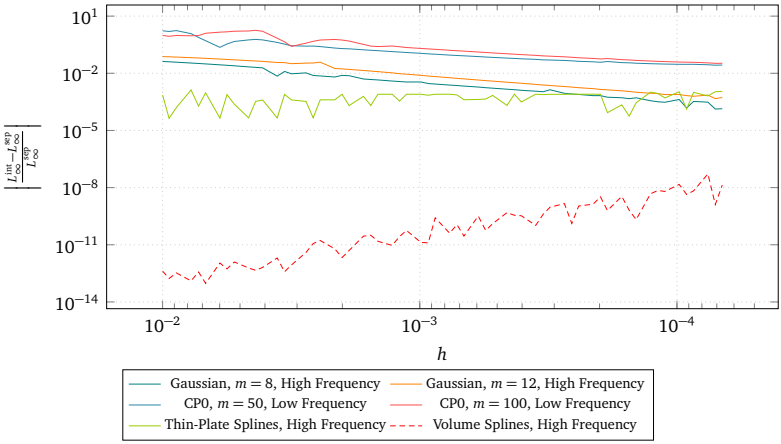


Figure 4.19: Relative difference of the L_∞-error of interpolations with integrated and separated polynomial for different basis and test functions.

the error saturation.

Comparing the Integrated and Separated Formulations In the introduction of this section, I stated that the RBF formulation with integrated Eq. (3.4) and the one with separated polynomial Eq. (3.9) yield almost identical results.

Figure 4.19 compares both implementations and shows the relative difference of the error between them. Even though the absolute value of the relative difference is shown to avoid negative values in a logarithmic scale, in the vast majority of samples, the separated polynomial implementation outperforms the integrated polynomial, when looking at the error. A notable exception to that is the interpolation of the constant function (not shown here). The interpolation with the integrated polynomial achieved an exact interpolation with $L_\infty = 0$, while the implementation using the separated polynomial only reached close-to-zero values of $L_\infty < 10^{-15}$.

As stated before, the separated polynomial does not change the interpolation or evaluation matrices and therefore C and A remain identical to the standard RBF interpolant without polynomial extensions. The integrated polynomial adds $2\,(1+d)\cdot|\Xi|$ entries to matrix C and $(1+d)\cdot|\Delta|$ entries to matrix A for an interpolation problem in d dimensions. These densely populated blocks added to the matrices hold the coordinates of the input or output mesh, respectively. This comes with some downsides affecting the computational properties. The

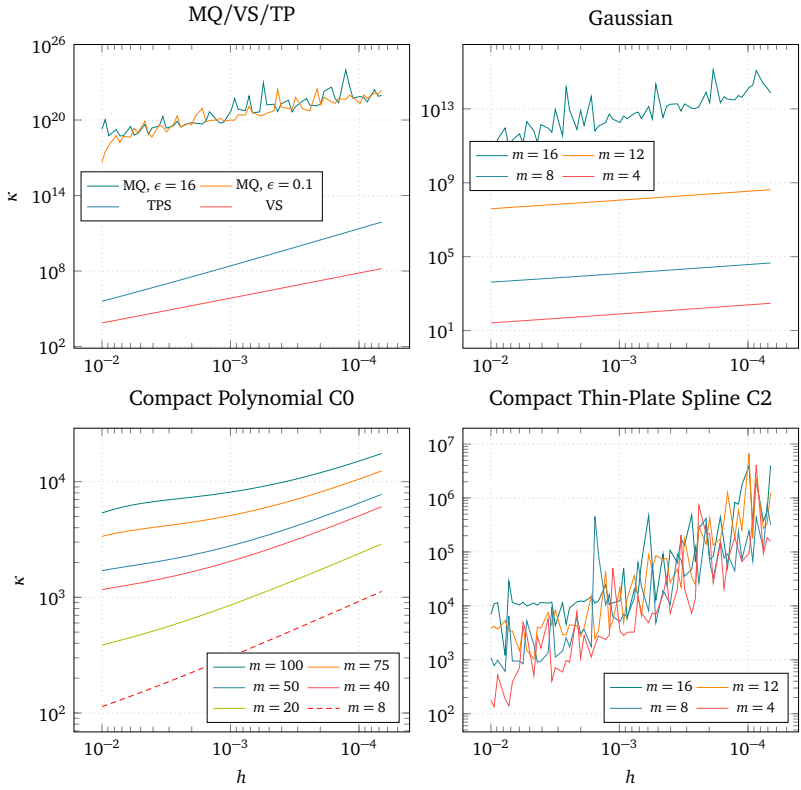


Figure 4.20: Condition number κ of the interpolation matrix C with integrated polynomial for different basis functions and shape parameters over the mesh width h.

Test Function	m	L_∞-error	RMSE	m_{opt}	L_∞-error at m_{opt}	RMSE at m_{opt}
Jump	8	1.2835	0.0363	4	0.9674	0.0236
Rosenbrock	8	25.8512	0.5584	16	0.1476	0.0176
Eggholder	8	64.7353	1.2000	5	70.9040	1.1051

Table 4.2: Interpolation results for two-dimensional basis functions with separated polynomial. The optimal value for m is based on the minimum RMSE.

row-wise decomposition used by PETSc and the fact that the rows holding the coordinates are all assigned to rank zero introduces some load imbalance to the problem. Depending on the selected (compact) basis function and shape parameter, the basis function values are close to zero for most of the vertices and therefore yield a sparse matrix. This sparsity can be exploited by numerical schemes and by using an appropriate storage format, which is potentially hindered by the dense blocks added. The symmetry of A and C, on the other hand, is not impaired by the integrated polynomial.

Figure 4.20 shows the condition number of the matrix C with integrated polynomial. To compare with variants without or with separated polynomial, refer to Fig. 4.10. The integrated polynomial causes a moderate increase of the condition number throughout all cases observed. In Fig. 4.10 it can be seen, that the compact basis functions, without or with separated polynomial show a constant condition number over the mesh size and appropriately scaled basis functions. The integrated polynomial, on the other hand, causes the condition number to increase steadily with h, leading to numerical instability at lower values of m and, thus, a lower level of accuracy that can be achieved.

Two-dimensional Interpolation Finally, the two-dimensional test functions are evaluated and compared to the results with the standard interpolant from Sec. 4.2. The visualization of the results is almost identical to Fig. 4.14 and for that reason won't be shown here. Also, most of the analysis there also applies to the interpolation with polynomial. The results are very similar and consistent with the data from one-dimensional interpolation. The polynomial ensured a moderate improvement in the RMSE and, apart from the Rosenbrock test function, also in the L_∞-error. While the test function values are not centered around $z = 0$, they are not far off. As demonstrated in Fig. 4.18, a shift of the function has a direct effect on the effectiveness of the polynomial.

Higher Degree Polynomials Previously in this section, the polynomial used always was of linear degree, i.e., $k = 2$. Here, I will briefly discuss the opportunities of extending

Eq. (2.33) with a polynomial of higher degree for one-dimensional test cases.

The implementation essentially works like the separated polynomial, in the sense that it subtracts the fitted polynomial from the input data before the core RBF interpolation takes place. Afterwards, the fitted polynomial is evaluated on the output mesh and added to the results. In contrast to the implementation described in Sec. 3.2, I do not explicitly create the Vandermonde matrices Q and V, but use an implementation of polynomial fitting from NumPy.

Figure 4.21 shows how using a polynomial of higher degrees effects the accuracy. Additionally to the test mesh of the same geometric dimensions as the input mesh, the error is also evaluated on a slightly larger interval of $[-0.1, 1.1]$ to explore a potential benefit on the interpolation quality outside the input domain.

The effect is strongly dependent on the test function used. The low-frequency test function profits most from higher degrees. A polynomial with degree 19 achieved an absolute improvement in the L_∞-error of 0.249 on the normal sized mesh and an absolute improvement of 12 on the larger mesh, which is a relative improvement of $1.4 \cdot 10^{13}$ on the normal mesh and $2.3 \cdot 10^{11}$ on the larger mesh. For the high-frequency test function, the optimal degree is only 3, with an absolute improvements in the L_∞-error of 0.2 and 8.6 or an relative improvement of 28.5 and 3.5, respectively. For the remaining test functions, higher degrees have a rather detrimental effect on accuracy. This is especially evident when looking at the jump function, where it can be attributed to the Runge phenomenon.

While these first numbers partly look promising at first, higher degree polynomials come with a number of disadvantages. The dependency on the test data, which is not known beforehand in simulations comes first. Even though the low-frequency test function contains exponential and trigonometric terms, it seems to "fit" the polynomial interpolation here and probably is a positive outlier. It is well-known that polynomial interpolation tends to produce heavy oscillations at higher orders, which eventually become harmful to the accuracy. Furthermore in a parallel algorithm, it introduces an amount of global communication increasing with higher polynomial degrees.

For these reasons, I decided to not pursue using polynomials of higher than degree one together with the RBF interpolation.

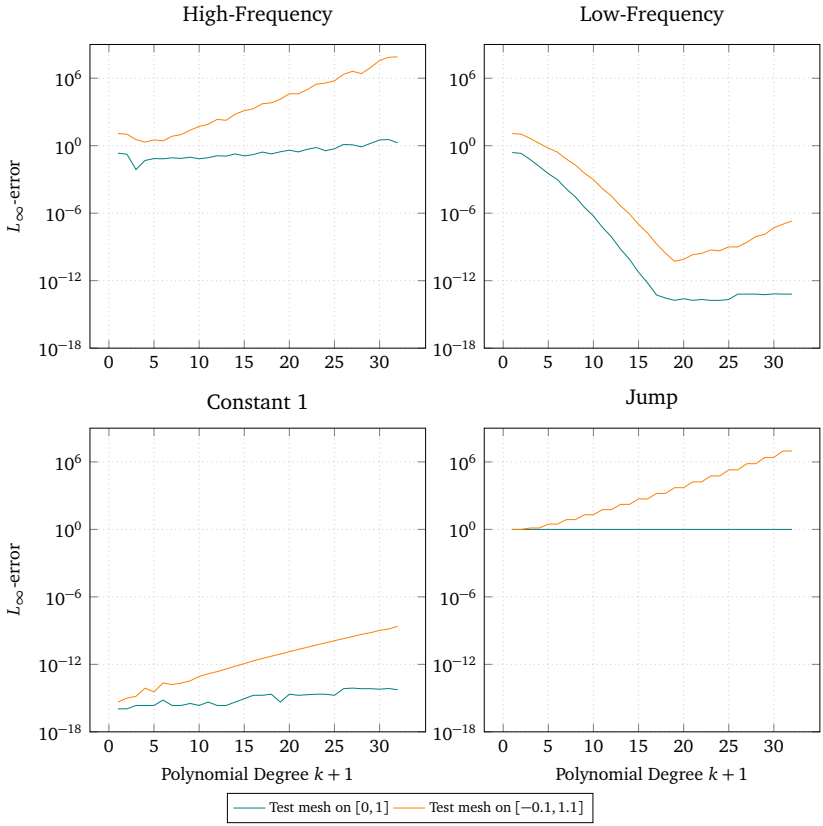


Figure 4.21: Error for RBF interpolation using a Gaussians ($m = 6$) basis together with polynomials of different degrees. The interpolant is constructed on $[0,1]$ and evaluated on two different test mesh.

4.4 Rescaled Interpolant

The method of rescaling the interpolant, as presented in Sections 2.4.6 and 3.3 is expected to mitigate the adverse effects of an insufficient density of basis functions. For a visualization of this effect, dubbed *hanging laundry*, due to its resemblance in appearance, see Fig. 2.6.

The rescaling is less a modification of the interpolation process but can rather be seen as a post-processing method. The interpolation matrix C and the evaluation matrix A are not modified, such that the condition number and stability remain unaffected.

The results of the rescaling procedure are quite diverse and vary with the basis function used. For the compact basis functions in Fig. 4.23, it shows a significant improvement of the results, especially for small support radii. For Gaussians, the results are shown in Fig. 4.22. In the stable regime, up to $m \approx 13$, the non-rescaled and no-polynomial variant performs worst. For the non-polynomial variant, the rescaling is able to achieve a constant improvement in the L_∞-error of about $2 \cdot 10^{-8}$ in the range $m = 7 \dots 13$. For smaller values of m, the improvement increases monotonically, to 0.6 for $m = 3$ and more. The comparison between the rescaled and non-rescaled variants with a polynomial shows a similar behavior, albeit less improvement due to the rescaling can be observed. At $m = 3$ the difference in error is 0.14, declining to $5 \cdot 10^{-9}$ in the same range of $m = 7 \dots 13$. In the unstable regime beyond $m = 13$, the rescaling still improves the results, though the instability makes an accurate interpolation impossible.

In Fig. 4.23, the results for the same setup using compact polynomial basis functions are shown. The general behavior is similar to the interpolation with Gaussian in the sense that the rescaling improves accuracy. As expected, the effect is most significant for smaller support radii and diminishes for larger values of m. The relative improvement by the rescaling show an asymptotic behavior, while the L_∞-error experiences error saturation.

The data from both compact basis functions, show that the accuracy of the rescaled versions with and without polynomials are error-wise on par. With rescaling, the effect of the polynomial is negligible. This has been confirmed for different basis functions, also variations that feature a large shift in y-direction. A notable exception to this is the constant test function, where the polynomial provides a significant advantage in the regime of small support radii, i.e., small values of m.

Global basis functions, i.e., volume splines and thin-plate splines remain largely unaffected by the rescaling and the polynomial. In Fig. 4.4, it can be seen that volume splines achieve an interpolation of the constant function with accuracy up to machine tolerance. Thin-plate splines also yield very accurate results on the constant function. Thus, the rescal-

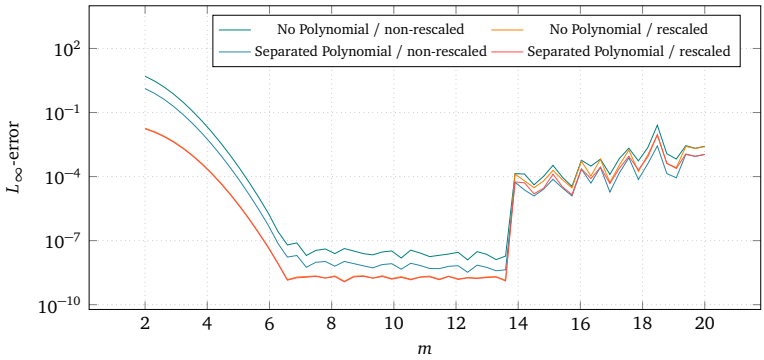


Figure 4.22: RBF interpolation of the high frequency test function using a Gaussian basis on $[0,1]$ with $h = 0.001$, evaluated on $[0.2, 0.8]$. The effect of rescaling for interpolation variants with and without polynomial is shown.

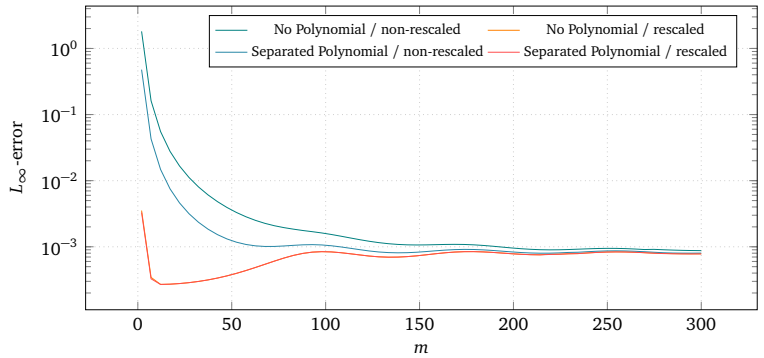


Figure 4.23: RBF interpolation of the high frequency test function using a Compact Polynomial C0 basis on $[0,1]$ with $h = 0.001$, evaluated on $[0.2, 0.8]$. The effect of rescaling for interpolation variants with and without polynomial is shown.

ing interpolant s_1 becomes almost 1 everywhere, making the rescaling ineffective.

The rescaling procedure adds some computational effort to the interpolation. It requires an additional solver step as the interpolant s_1 of the constant function $g(x) = 1$ needs to be computed. This solve is only required in the *offline* phase of the interpolation, i.e., once at the beginning. Furthermore, for solving the actual interpolant $C^{-1} f_\xi = \lambda$, information from solving the rescaled interpolant can be reused. This potentially includes the preconditioner of C and factorizations.

4.5 Adaptive Shape Parameters

So far, the shape parameter is chosen based on the mesh density as given in Eq. (3.2) for Gaussians or Eq. (3.3) for non-Gaussian, compact basis functions. The shape parameter is calculated using the maximum mesh width $h_{\max}$, resulting in a basis function support which includes a *minimum* of m vertices in every direction in the support radius for any basis function on a given mesh. In Sec. 4.2 and, furthermore, in Figs. 4.5, 4.6, 4.16 and 4.17, it is shown that a too large support radius can lead to unstable interpolation results.

The obvious remedy to this problem is to use a non-constant shape parameter, i.e., the basis function φ_i, $i \leq |\Xi|$ evaluated at each $\xi \in \Xi$ uses an individual shape parameter ϵ_i such that always the nearest m vertices of ξ in every direction are included in the support radius. In Sec. 2.4.4, this method is detailed further.

This method comes with a number of potential problems:

- Using a non-constant shape parameter ϵ destroys the symmetry of the matrices A and C as $\varphi_i(|\xi - \zeta|) \neq \varphi_j(|\zeta - \xi|)$ for $i \neq j$. This impairs the numerical properties, e.g., the matrices are not guaranteed to be non-singular anymore.

- The vector of shape parameters $\boldsymbol{\epsilon} \in \mathbb{R}^{|\Xi|}$ has to be retained in order to construct A. For the *online*-phase, i.e., for the subsequent coupling iterations, $\boldsymbol{\epsilon}$ is not required anymore.

- A definition of the local mesh width must be available. While this is trivial for one-dimensional meshes, it becomes exceedingly harder for meshes of higher dimensionality. From a theoretical point of view, finding appropriate definitions of a mesh width in high dimensions is non-trivial. From the practical point of view, information about neighborship in not generally available, i.e., it must be computed beforehand, which is computationally expensive, though spatial trees can be used for more efficiency, see Sec. 3.4.4.

For those reasons, the method is currently not implemented in preCICE. Nevertheless, I will evaluate it here to explore chances for future inclusion into preCICE.

Based on a global value for m, a basis function is created for each vertex ξ of a one-dimensional, non-uniform, ordered mesh Ξ, based on a local mesh width h_i. I define the local mesh width as

$$h_i = \frac{\Xi_{i-1} - \Xi_{i+1}}{2}. \tag{4.18}$$

The local shape parameter is then computed according to the usual definitions, Eq. (3.2) or Eq. (3.3).

Non-adaptive basis functions use $h_{\max}$ to scale the basis functions. That means, they are adjusted to high accuracy at the expense of stability in regions of lower mesh width, i.e., where $h_i \ll h_{\max}$. The support radius of the adaptive basis functions will therefore be smaller in such regions than in the non-adaptive case. Adaptiveness, as it is employed here, works as a measure to improve stability, not accuracy for a specific value of m. In the stable interpolation regime, the accuracy of adaptive interpolation with thus only be worse compared to non-adaptive interpolation. On the other hand, a stable interpolation can be achieved with an adaptive shape parameter in regions which would otherwise be unstable.

As a first test case, a mesh with continuously increasing mesh width is created, which $\xi_i = \xi_{i-1} + 0.001\,i$, resulting in 359 vertices on $[0, 1]$ with $h_{\max}/h_{\min} = 4.6$. The results for Gaussian basis functions are shown in Fig. 4.24.

First results show an improvement regarding the error of the interpolation. At $m = 8$ and $m = 10$, the non-adaptive interpolation shows instability at the fine parts of the mesh up to $x = 45$ or $x = 70$, respectively. The adaptive interpolation avoided this instability by using basis functions with a smaller support radius. As the mesh width approaches $h_{\max}$, the adaptive interpolation convergences to the non-adaptive. The slight increase in error on the right side of the mesh probably indicates the beginning of instability. Boundary effects are excluded by evaluating only an appropriately padded subset of the input mesh.

The adaptive interpolation comes with a significant reduction of the condition number of matrix C by up to 9 orders of magnitude for $m = 8$ and still 3 orders of magnitude for $m = 30$.

Figure 4.25 compares adaptive with non-adaptive interpolation on a mesh with a Gauss-Chebyshev point distribution for increasing underlying polynomial orders. In the non-adaptive plot, the region of stability can clearly be discerned from the unstable region. For small support radii, such as $m = 4$, instability begins at higher orders. It sets in earlier as the support radius m grows larger. Even though $h_{\max}$ is computed for each Gauss-Chebyshev distribution of a certain order p, since higher order does not only result in a

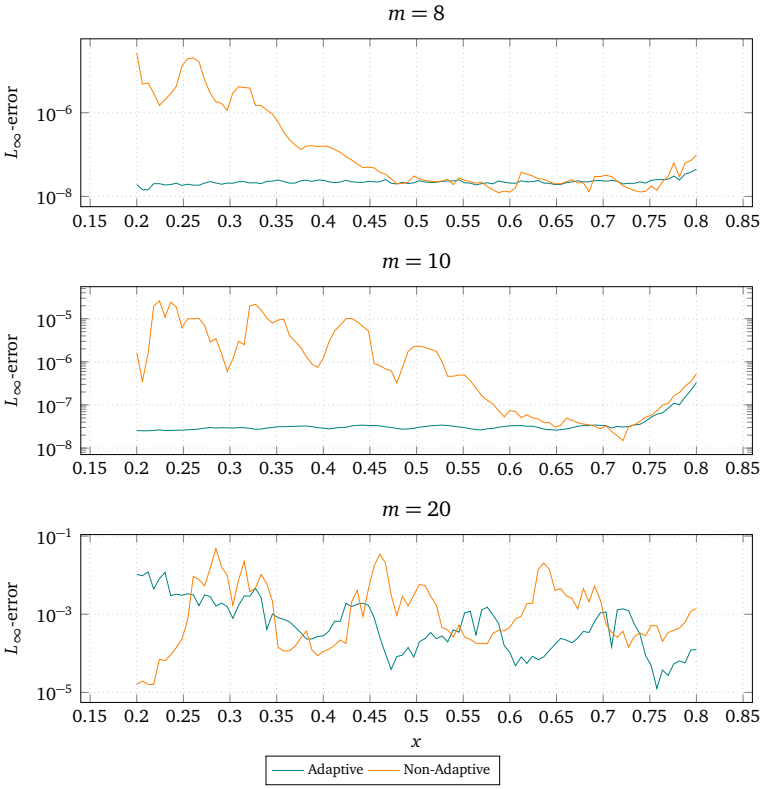


Figure 4.24: Adaptive and non-adaptive interpolation using Gaussian basis function on a mesh with continuously increasing mesh width. The input data are generated from the high frequency test function. To smoothen the highly oscillatory error, 5000 samples on $[0.2, 0.8]$ are aggregated to 100 data points by taking the maximum of each chunk of 50 samples.

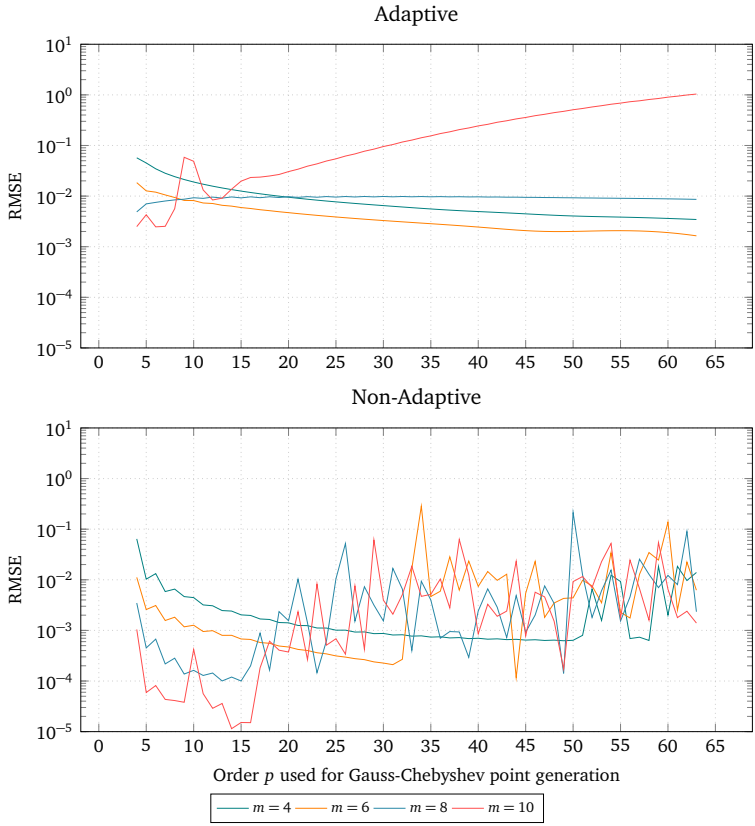


Figure 4.25: Adaptive and non-adaptive interpolation using Gaussian basis functions on a mesh with a Gauss-Chebyshev point distribution. The mesh consists of 16 cells of size 0.0625 with polynomial order in the range $p = 4 \ldots 65$. The test mesh's spatial extension is identical to the input mesh. To not overstate boundary effects, the root mean-squared error is used here instead of the L_∞-error.

finer mesh resolution, but also in a larger mesh width inequality $h_{\max}/h_{\min} \approx p/2$, high orders will inevitably lead to disproportionately scaled basis functions and with that to unstable interpolation. The adaptive interpolation is much less affected by the order of the mesh, resulting in a consistently stable interpolation for sufficiently small support radii. Still, the correct choice of m is critical, as can be seen from the result for $m = 10$.

It should be noted, that the used definition of mesh width Eq. (4.18) does not guarantee that a particular basis function φ_i and shape parameter ϵ_i exactly include the desired number m of neighboring vertices in every direction. Instead, it provides an approximation of the local mesh width, based on the two next neighbors. The quality of this approximation will deteriorate with higher orders, as the mesh width inequality increases.

4.6 RBF-QR Basis Transformation

The RBF-QR algorithm uses a transformation of the standard RBF basis in order to achieve a system matrix with improved properties. It is briefly introduced in Sec. 2.4.2. The mathematical theory and a Matlab implementation can be found in [45]. Here, the Python implementation, based on the work in [97] is used.

The RBF-QR algorithm uses a series expansion of the basis function, which must be constructed manually for each dimensionality and basis function. In [45], expansions for Gaussian and Bessel functions in up to three dimensions are presented. Here, I will focus on the Gaussian basis function in one dimension. Furthermore, as of today, no conservative formulation of the RBF-QR algorithm exists.

For brevity, I will denote the standard RBF algorithm with separated polynomial as RBF-P in this section.

The algorithm aims to be stable, even in the limit of increasingly flat basis functions, i.e., as $\epsilon \to 0$. Figure 4.26 shows RBF-QR and RBF-P using a shape parameter of $\epsilon = 0.0004$. Even smaller values for ϵ make the matrix C of the RBF-P implementation become singular. The given shape parameter results in a support radius of approximately 11 381, converging to the so-called flat limit of Gaussian basis functions. The figure clearly shows that the theoretical gain in accuracy of increasingly flat basis function can not be exploited due to instabilities in the RBF-P algorithm. RBF-QR, on the other hand, achieves low error values even for small shape parameter, i.e., large support radii.

For the numerical experiments, I therefore use a small, constant shape parameter of $\epsilon = 10^{-5}$ for RBF-QR and $m = 6$ for RBF-P.

Figure 4.27 compares RBF-P and RBF-QR interpolation of the high frequency test function

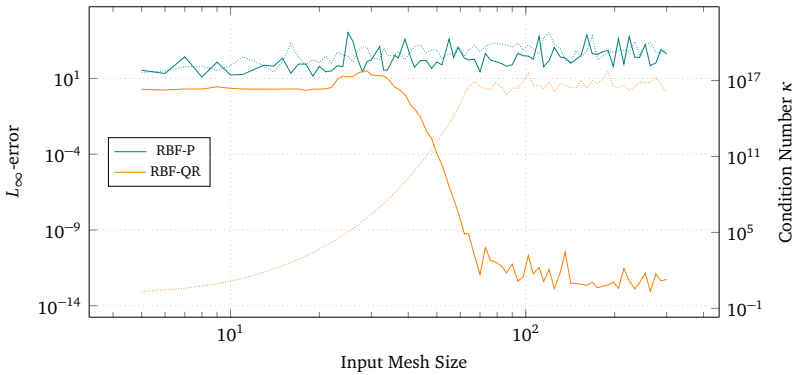


Figure 4.26: Error and condition number for the RBF-QR and RBF with separated polynomial (RBF-P) algorithms. The high frequency test function is used. The RBF-P algorithm uses a Gaussian basis function. RBF-QR and RBF-P use the same support radius based on a shape parameter of $\epsilon = 0.0004$. The error is evaluated on $[0.2, 0.8]$ to avoid boundary effects. Solid lines indicate the error, dotted lines the condition number.

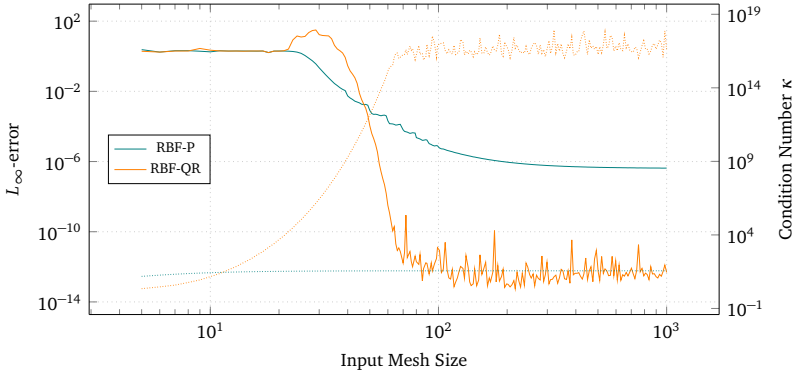


Figure 4.27: Error and condition number for the RBF-QR and RBF with separated polynomial (RBF-P) algorithms. The high frequency test function is used. The RBF-P algorithm uses Gaussian basis functions with $m = 6$, RBF-QR uses a constant shape parameter of $\epsilon = 10^{-5}$. The error is evaluated on $[0.2, 0.8]$ to avoid boundary effects. Solid lines indicate the error, dotted lines the condition number.

on a uniform mesh. For small mesh sizes both yield a similar accuracy. The RBF-P accuracy improves until it reaches a stable level due to the effect of error saturation (cf. Sec. 2.3.3). The RBF-QR implementation quickly falls to a low error of around 10^{-12}, where is stagnates with oscillations. These oscillations are also reflected in the condition number. While κ from matrix C of the RBF-P algorithm remains constant due to scaling of the basis functions, RBF-QR peaks at $\kappa = 10^{18}$ in the unstable regime. Experiments with much larger mesh sizes show that this trend continues.

The interpolation on non-uniform meshes of Gauss-Chebyshev type shows diverse results. In Fig. 4.28 (top) promising results for a single cell are shown. The condition number remains constant at a low level for increasing underlying polynomial order p, in contrast to the steady increase of κ of the RBF-P. Note that the increase in the condition number does not originate from the large mesh as such, but from the increasing non-uniformity with increasing order. The error for higher orders $p > 75$ falls to values in the order the machine precision, whereas the RBF-P is not able to achieve an accuracy better than 10^{-3}. An explanation to this almost optimal performance on a single Gauss-Chebyshev cell can be found in [45, Sect. 4.1.2], where the expansion of the Gaussian basis function is done in a Chebyshev basis. Thus, the RBF-QR is essentially performing polynomial interpolation with Chebyshev polynomials on Gauss-Chebyshev points.

When looking at more than one cell of Gauss-Chebyshev type, shown in the bottom of Fig. 4.28, the results are less promising. In further experiments, the RBF-QR algorithm shows that towards the boundaries of the interpolation region, the quality quickly deteriorates with heavy oscillations in the interpolant. These oscillation at the boundaries also dominate the error in the data shown. If the domain of the test mesh is limited to $[-0.6, 0.6]$, it can be shown that the results of the RBF-QR improve tremendously, e.g., for an order of 100, the L_∞-error improves from 32 to $2.8 \cdot 10^{-5}$ when restricting the error measurement on $[-0.6, 0.6]$.

This experimental data show that, while the RBF-QR interpolation is not susceptible to the increase of the condition number as $\epsilon \to 0$, it is sensitive to an increasing mesh size, especially for uniform meshes. This sensitivity, however, is on a similar level as for the standard RBF-P interpolation with constant m.

The more problematic trait of the RBF-QR is the extreme loss of accuracy near the boundary. While this is also present with standard RBF interpolation, it is much more pronounced in the RBF-QR results. However, reducing the size of the output, relative to the input domain is usually not an option in standard coupling scenarios.

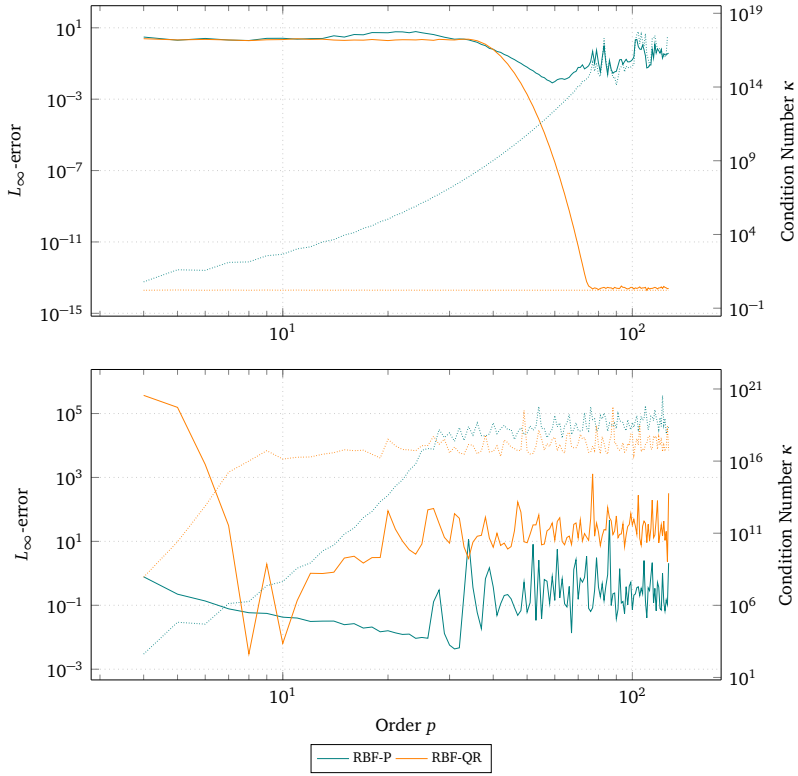


Figure 4.28: Error and condition number for the RBF-QR and RBF with separated polynomial (RBF-P) algorithms on a mesh on $[-1, 1]$ with Gauss-Chebyshev type point distribution of order up to 128. The top plot show results for increasing underlying polynomial order for a single cell, the bottom plot for 8 cells. The high frequency test function is used. The RBF-P algorithm uses Gaussian basis functions with $m = 6$, RBF-QR uses a constant shape parameter of $\epsilon = 10^{-5}$. The error is evaluated on $[-0.9, 0.9]$. Solid lines indicate the error, dotted lines the condition number.

4.7 Conservative Interpolation

The algorithms evaluated in this chapter so far are the consistent variants of the radial-basis function interpolation. In this section, I will focus on the conservative formulation. For a general introduction of the differences between consistent and conservative interpolation, see Sec. 2.1 and Sec. 3.2.2. In Sec. 4.1.2, some of the problems regarding the evaluation of conservative interpolation methods are detailed. For consistent interpolation, the output mesh has no direct influence on the interpolation, as long as, for measurements, a sufficiently fine output mesh is used to ensure a valid evaluation of the error.

Conservative interpolation needs to fulfil the conservation condition $\sum_{\Xi} \xi_i = \sum_{\Delta} s(\delta_i)$, which rules out $\|s(\delta_i) - f(\delta_i)\|$ as a informative error metric. I will therefore not only evaluate the usual error metrics, i.e., RMSE and L_∞-error, but also use the alternative methods presented in Sec. 4.1.2.

I show results for the high frequency test function and increasing output mesh resolutions in Fig. 4.29. The input mesh is perturbed by a random value in the range of $\pm 0.15\,h$ to avoid identical meshes. In contrast to consistent interpolation, conservative interpolation constructs the matrix C from the output mesh. For that reason the Gaussian and compact polynomial basis functions are scaled relative to the output mesh.

Comparing $\sum_{\Xi} \xi_i$ and $\sum_{\Delta} s(\delta_i)$ gives a measure how far the conservation property is violated. I call this value the *conservative delta*. In all cases, the difference between the values remain in the range of 10^{-13} and, therefore, conservation is fulfilled up to a very small error.

In the present setup, the input mesh size is fixed, while the output mesh size is varied. Reversing that situation, i.e., fixed output mesh and variable input mesh, gives the experimental setup similar to the one used for the evaluation of consistent interpolation. The results of this setup follow the trends seen in Fig. 4.29. Conservativness is fulfilled in all cases, as well.

Figure 4.30 illustrates results for varying output mesh sizes and with this shows the effect of the conservativity on the output. It shows the interpolation of the high frequency test function from an input mesh of 300 nodes to 250, 300 and 350 nodes. The corresponding results are given in Tab. 4.3. The different mesh sizes on the output mesh cause a shift up- or downwards, respectively. This shift is almost linear, with a weak dependency on the value of the input data. From this figure, we can easily see, why the rescaled error is the suitable metric for conservative mapping: It does not correlate with the inevitable offset, caused by different numbers of nodes in the input and output meshes. Furthermore, it was

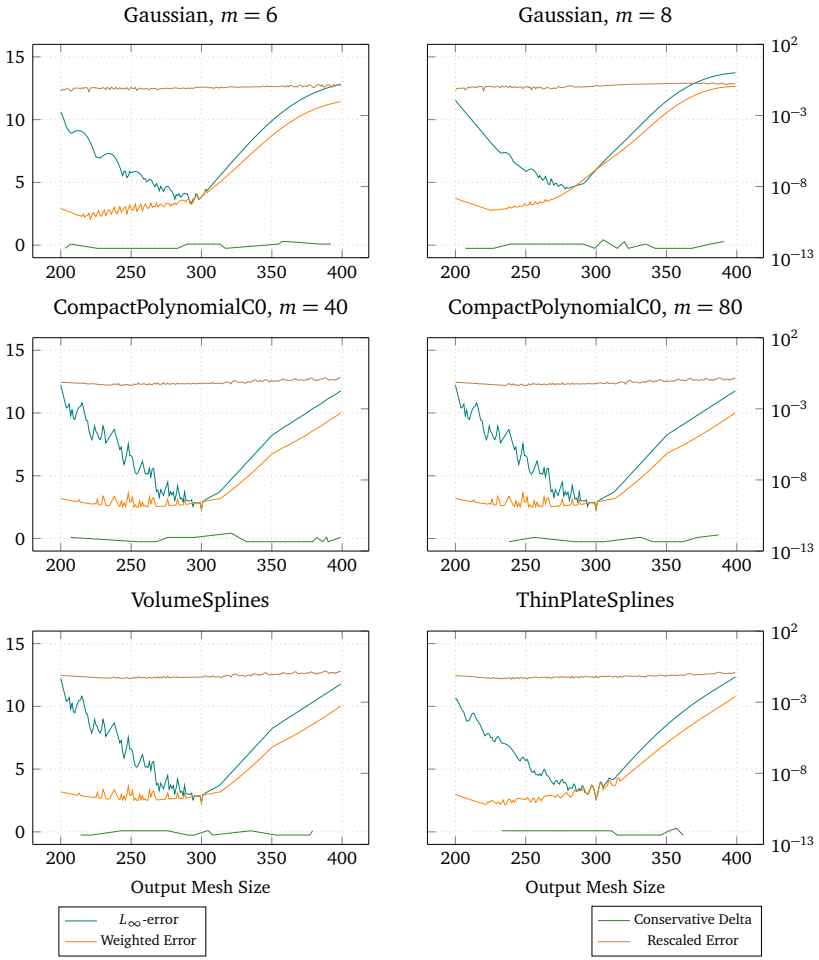


Figure 4.29: Conservative Interpolation using RBFs with separated polynomial on the high frequency test function. The input mesh consists of 300 vertices that are shifted by a random value between $\pm 0.15\,h$ to avoid a perfect match between input and output mesh. The conservative delta is the difference of the sum of all values on each mesh. Note that the line for *conservative delta* is broken where the results are numerically 0.

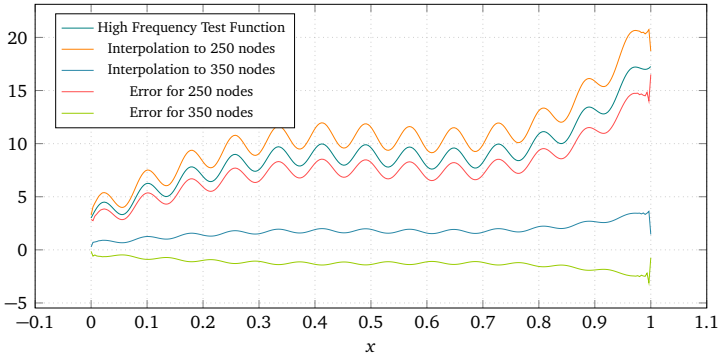


Figure 4.30: Conservative interpolation with Gaussians and separated polynomial of the high frequency test function. Shown are interpolation results for output meshes of different sizes and the pointwise difference to the test function.

	250 nodes	300 nodes	350 nodes
L_∞-Error	4.0606	$4.9738 \cdot 10^{-14}$	3.3315
Weighted Error	1.6638	$4.9738 \cdot 10^{-14}$	2.0055
Rescaled Error	0.0162	$7.8160 \cdot 10^{-14}$	0.0202
Conservative Delta	$-4.5475 \cdot 10^{-13}$	$-1.3642 \cdot 10^{-12}$	$1.3642 \cdot 10^{-12}$

Table 4.3: Results for conservative interpolation with a Gaussian basis from 300 nodes to 250, 300 and 350 nodes.

tested with tainted data (e.g. comparing against a different test function than the target function for the interpolation) and in situations where an accurate interpolation can not be achieved (e.g. basis function with a very small support radius). In these situations, the *rescaled error* shows to appropriately reflect the artificial error. The conservative delta, on the other hand, is only a global metric and does not reflect local effects in the interpolation domain. It is always almost zero, even in interpolation conditions, where an error measure should be significantly higher. For a point-wise comparison of the different error metrics, see Fig. 4.3.

As both the *weighted error* and the *rescaled error* are pointwise error definitions, the L_∞-norm is applied.

Conservative interpolation brings an extra level of complexity into the interpolation problem. The stability and accuracy not only depend on the input mesh and the selected basis function, as it is the case with consistent interpolation, but also on the output mesh.

Even when the input mesh is held constant, instability can occur. For basis functions that have a parameterized support radius, such as Gaussians or compact polynomials, scaling the function according to the input mesh, as opposed to the output mesh produces almost identical results and hence no improvement in the stability range.

4.8 Runtime Characteristics of PETSc Linear Solvers

This section explores how the runtime characteristics of the PETSc RBF implementation varies with changes to the interpolation parameters. I will observe the global time to solution, the time for the linear solver and the number of iterations for different combinations of basis functions, shape parameters, linear solvers and preconditioners.

4.8.1 Test Setup

In contrast to the previous tests, I do not use the Python implementation of the RBF algorithm, but the parallel C++ implementation from preCICE. The input mesh is two dimensional $[0,1] \times [0,1]$ with 100^2 elements in total. This results in a mesh width of $h = 1/100$. The input data are generated as $f_\xi = i + j + 1$, with $i, j = 1 \dots 100$ being the vertex indices. The output mesh has no influence on the interpolation matrix C, it is simply chosen identical.

The solvers are evaluated on three different basis functions. Gaussians, with different shape parameters, ranging from $m = 10$ up to $m = 100$, as an example of a basis function that is defined on a local support, thus yielding sparse matrices. It is compared to thin-plate splines and volume splines, which both have a global support.

Both participants are executed on two processors each.

I limit the selection of solvers to the ones included in the PETSc default installation. The selection of applicable solvers is further limited by the type of the matrix C, which is `MATSBAIJ` for symmetric block sparse matrices. Furthermore, the solver must exist in a parallel version.

The *Generalized Minimal Residual* method (GMRES) as given in [88] is the default solver from PETSc. It works for non-symmetric matrices and is known to generally perform well. The *Biconjugate Gradient method* (BiCG) [44] is a variant of the conjugate gradient method, which also works on non-symmetric and indefinite systems. A stabilized variant of the BiCG method, the *Biconjugate Gradient Stabilized method* (BiCG-stab) [106] is also tested.

All of the aforementioned methods are Krylov subspace methods. This family of methods

usually requires a preconditioning in order to reach good convergence rates. In PETSc, preconditioners are implemented as a decomposition algorithm and the actual preconditioner, which works only on the local block of the matrix.

For decomposition, I evaluate the *Restricted Additive Schwarz* [36] (asm) and the *Block Jacobi* (bjacobi) methods. As the inner, local preconditioner, an *Incomplete Cholesky Factorization* [23] (icc) is used.

Convergence or divergence is checked for against a number of criteria. Let $Ax = b$ be the linear system to be solved and $r_k = b - Ax_k$ the residual at iteration k. As laid out in [3, p. 90], the convergence (or divergence) is determined by PETSc using three quantities, a limiter for the residual norm relative to the norm of the right hand side t_r, a threshold for the absolute size of the residual norm t_a and a divergence limit for the relative residual norm t_d. Convergence is detected if

$$\|r_k\|_2 < \max(t_r \cdot \|b\|_2, t_a). \tag{4.19}$$

Divergence is detected if

$$\|r_k\|_2 > t_d \cdot \|b\|_2. \tag{4.20}$$

Divergence according to Eq. (4.20) is marked by `DIV DTOL` in the following. The maximum number of solver iterations is set to 20000, twice the default. A solver run that fails due to hitting the iteration bound is marked by `DIV ITS`. Furthermore, `DIV BREAK` indicates a general breakdown in the Krylov method, likely due to a singular matrix or preconditioner.

4.8.2 Results

The results of interpolation using Gaussian basis functions with different shape parameters are listed in Tables 4.5 and 4.6. Without surprise, the time per iteration increases with the support radius, i.e., decreasing ϵ or increasing the number of non-zeros entries in the matrix.

Table 4.4 lists the results for basis functions of the types thin-plate splines and volume splines. Both are defined on a global support and, thus, produce dense interpolation matrices, resulting in a significantly lower iteration speed than with Gaussians. For volume splines, the linear solver is able to achieve convergence in more cases than with thin-plate splines.

Comparing the integrated and the separated formulations of the polynomial for thin-plate splines yields that, for the integrated polynomial only the stabilized BiCG method is

not able to achieve convergence. The picture for volume-splines is less pronounced, but alike showing the fact, that, for the linear system from the separated RBF formulation, it is significantly harder to achieve convergence.

For almost all combinations of basis function, solver and preconditioner, it turns out that the separated polynomial has an negative effect on the convergence rate. Frequently, the variant with the separate polynomial is not able to achieve convergence within the maximum number of iterations. For Gaussians, only with the integrated polynomial, convergence is achieved for larger support radii. Combinations where the separate polynomial requires less iterations to achieve convergence, such as volume splines with GMRES and bjacobi/icc preconditioner can not be generalized to, e.g., other basis functions.

The integrated polynomial adds $d+1$, d being the dimensionality of the problem, densely populated rows and columns to the matrix C. The additional communication effort due to the additional entries is reflected in a lower number of iterations per second. However, the overall time to solution is clearly dominated by the number of iterations, not by the time per iteration.

Due to the high number of parameters of the test setup, an exhaustive comparison of all possible parameter combinations is clearly out of reach. From the available data, it is hard to draw conclusions and to give a definitive recommendation for an optimal solver. For Gaussians, especially using the integrated polynomial, the GMRES solver with asm/icc preconditioner stands out, being the only one achieving convergence at larger support radii. For thin-plate splines and volume splines, the BiCG solver with asm/icc preconditioner seems to be a good choice.

BF	Polynomial	Solver	PC	Iterations	T_G [s]	T_S [s]	$^{\text{Its}}/_{T_S}$ [s^{-1}]
TPS	integrated	bcgs	asm/icc	DIV BREAK	–	–	–
TPS	separate	bcgs	asm/icc	DIV DTOL	–	–	–
TPS	integrated	bcgs	bjacobi/icc	594	339	194	3
TPS	separate	bcgs	bjacobi/icc	DIV ITS	–	–	–
TPS	integrated	bicg	asm/icc	831	715	571	1
TPS	separate	bicg	asm/icc	6453	4935	2756	2
TPS	integrated	bicg	bjacobi/icc	867	404	261	3
TPS	separate	bicg	bjacobi/icc	10610	3793	2375	4
TPS	integrated	gmres	asm/icc	1284	624	482	3
TPS	separate	gmres	asm/icc	DIV ITS	–	–	–
TPS	integrated	gmres	bjacobi/icc	498	262	119	4
TPS	separate	gmres	bjacobi/icc	DIV ITS	–	–	–
VS	integrated	bcgs	asm/icc	DIV DTOL	–	–	–
VS	separate	bcgs	asm/icc	2034	3118	876	2
VS	integrated	bcgs	bjacobi/icc	689	317	175	4
VS	separate	bcgs	bjacobi/icc	1223	2456	266	5
VS	integrated	bicg	asm/icc	645	633	491	1
VS	separate	bicg	asm/icc	1019	1112	435	2
VS	integrated	bicg	bjacobi/icc	874	364	224	4
VS	separate	bicg	bjacobi/icc	1287	675	294	4
VS	integrated	gmres	asm/icc	3016	992	854	4
VS	separate	gmres	asm/icc	2729	1342	603	5
VS	integrated	gmres	bjacobi/icc	2481	441	301	8
VS	separate	gmres	bjacobi/icc	3666	836	409	9

Table 4.4: Behavior of different linear solver and preconditioner combinations for thin-plate splines and volume splines. T_G denotes the global time for the entire interpolation, T_S the time for the linear solver only. "–" indicates that no solution is achieved.

BF	Polynomial	ϵ	Solver	PC	Iterations	T_G [s]	T_S [s]	$^{\text{Its}}/_{T_S}$ [s^{-1}]
GS	separate	207.23	gmres	bjacobi/icc	3	0	0	3000
GS	separate	103.62	gmres	bjacobi/icc	4	0	0	1333
GS	separate	69.08	gmres	bjacobi/icc	8	1	0	667
GS	integrated	51.81	bicg	bjacobi/icc	722	4	3	216
GS	separate	51.81	gmres	asm/icc	7	1	0	318
GS	separate	51.81	gmres	bjacobi/icc	14	1	0	400
GS	separate	46.05	bcgs	asm/icc	111	3	1	160
GS	separate	46.05	bcgs	bjacobi/icc	6246	91	35	179
GS	separate	46.05	bicg	asm/icc	78	2	0	160
GS	integrated	46.05	bicg	bjacobi/icc	DIV ITS	–	–	–
GS	separate	46.05	bicg	bjacobi/icc	265	4	1	179
GS	integrated	46.05	gmres	asm/icc	321	3	2	200
GS	separate	46.05	gmres	asm/icc	145	2	0	303
GS	integrated	46.05	gmres	bjacobi/icc	1050	4	3	330
GS	separate	46.05	gmres	bjacobi/icc	DIV ITS	–	–	–
GS	integrated	41.45	bcgs	asm/icc	237	4	3	89
GS	separate	41.45	bcgs	asm/icc	DIV DTOL	–	–	–
GS	integrated	41.45	bcgs	bjacobi/icc	1071	9	7	145
GS	separate	41.45	bcgs	bjacobi/icc	DIV DTOL	–	–	–
GS	separate	41.45	bicg	asm/icc	DIV ITS	–	–	–
GS	integrated	41.45	bicg	bjacobi/icc	9247	65	63	146
GS	separate	41.45	bicg	bjacobi/icc	DIV ITS	–	–	–
GS	integrated	41.45	gmres	asm/icc	421	4	2	169
GS	separate	41.45	gmres	asm/icc	DIV ITS	–	–	–
GS	integrated	41.45	gmres	bjacobi/icc	DIV ITS	–	–	–
GS	separate	41.45	gmres	bjacobi/icc	DIV ITS	–	–	–

Table 4.5: Behavior of different linear solver and preconditioner combinations for Gaussian basis functions and shape parameters $\epsilon = 207.23 \ldots 41.45$. T_G denotes the global time for the entire interpolation, T_S the time for the linear solver only. "–" indicates that no solution is achieved.

BF	Polynomial	ϵ	Solver	PC	Iterations	T_G [s]	T_S [s]	$^{\text{Its}}/_{T_S}$ [s^{-1}]
GS	integrated	37.68	bicg	asm/icc	779	11	9	84
GS	separate	37.68	bicg	asm/icc	DIV ITS	–	–	–
GS	integrated	37.68	bicg	bjacobi/icc	13043	111	110	118
GS	integrated	37.68	gmres	asm/icc	540	6	4	145
GS	integrated	37.68	gmres	bjacobi/icc	DIV ITS	–	–	–
GS	integrated	34.54	bicg	bjacobi/icc	16061	159	156	103
GS	separate	34.54	gmres	asm/icc	DIV ITS	–	–	–
GS	integrated	31.89	bicg	bjacobi/icc	16364	190	188	87
GS	integrated	27.63	bicg	bjacobi/icc	15605	235	231	67
GS	integrated	24.38	bicg	bjacobi/icc	12737	238	233	55
GS	integrated	20.72	bcgs	asm/icc	454	27	19	23
GS	integrated	20.72	bcgs	bjacobi/icc	3050	80	73	42
GS	integrated	20.72	bicg	asm/icc	1655	71	64	26
GS	integrated	20.72	bicg	bjacobi/icc	15517	388	381	41
GS	integrated	20.72	gmres	asm/icc	2754	62	55	50
GS	separate	20.72	gmres	asm/icc	DIV ITS	–	–	–
GS	integrated	20.72	gmres	bjacobi/icc	DIV ITS	–	–	–

Table 4.6: Behavior of different linear solver and preconditioner combinations for Gaussian basis functions and shape parameters $\epsilon = 37.68 \ldots 20.72$. T_G denotes the global time for the entire interpolation, T_S the time for the linear solver only. "–" indicates that no solution is achieved.

4.9 Upscaling of the RBF Implementation

In this section, I will present the results of upscaling runs of the C++ implementation of radial-basis function interpolation in preCICE. The runs are performed using ASTE, which acts as an artificial minimal surrogate solver to preCICE, see Sec. 6.2.3. As mentioned previously, for the linear algebra heavy-lifting, i.e., management of distributed matrix data structures and solving of linear systems, PETSc is used.

The input and output meshes are created on the unit square $[0,1]^2$ with the same number n of mesh vertices in x- and y-direction. The mapping computation is performed on participant B. For that reason, only this participant is upscaled, whereas participant A runs on a constant number of processors. The size of the matrix C and, in this case also of A, is $n^2 \times n^2$. This, however, does not reflect the actual number of non-zero elements, which is much lower, as detailed below. The exponential growth of the matrix size needs to be taken into account when analysing the upscaling results.

The values on the source mesh are set according to the function $f(x,y,z) = \sin(x)\cdot\exp(z)$. The meshes are decomposed according to the target number of processors using the *uniform* decomposition method, see Sec. 6.2.3.

4.9.1 Offline-Phase

In this section I will focus on the phase of the algorithm which needs to be computed only once for a simulation run, unless the mesh changes.

In Fig. 4.31, a qualitative comparison of the runtime contributions of different stages of the RBF algorithm is shown. In a nutshell, the principal stages are

Preallocation computes the number and values of the entries, i.e., evaluates the basis function and saves the number of non-zero entries per row, compare Sec. 3.4.4

Filling feeds the already computed entries into the PETSc data structures.

Post Filling assembles the PETSc matrices, e.g., distributes entries that are set off-rank to the correct rank.

Compute Rescaling solves C with a right-hand side of 1 for the rescaling interpolant. Depending on the linear solver, this phase already does a significant part of the actual solving of the interpolation system, such as factorization for preconditioners.

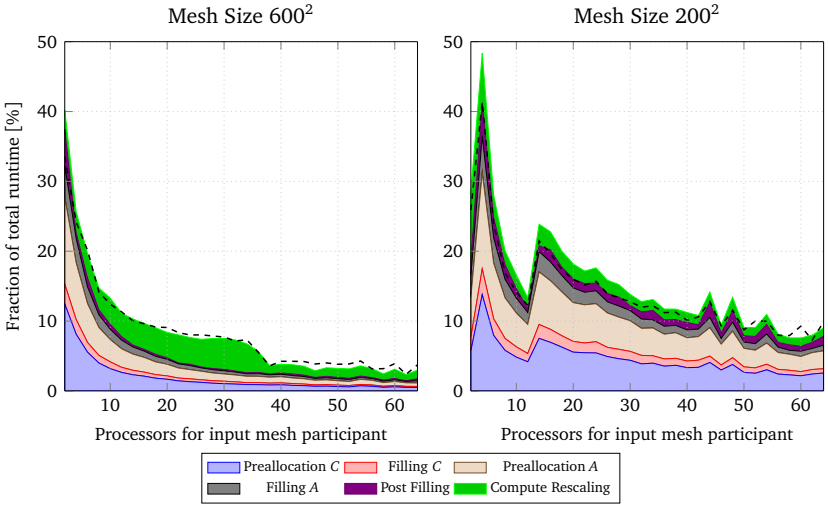


Figure 4.31: Relative runtime fractions of the stages of the RBF algorithm in a strong upscaling run for two different mesh sizes with Gaussian ($m = 6$) basis functions. 100% is the total runtime and also includes times not related to the mapping. The dashed plot line represents the percentage of the overall function `computeMapping`. Note that it may be slightly higher than the sum of its parts, as the times shown are the maximum overall ranks. The measurements were made on the Neon machine.

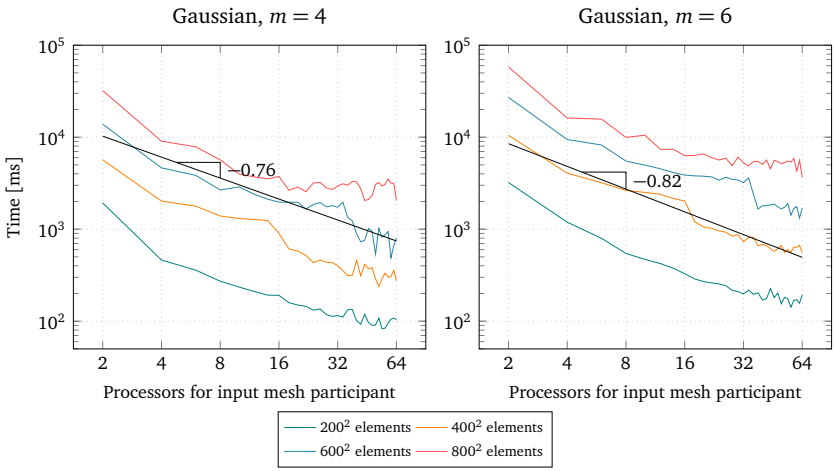


Figure 4.32: Strong upscaling on different mesh sizes with Gaussian basis functions on up to 64 processors. The time spent in the `computeMapping` event is shown, i.e., the entire offline phase of the mapping. Additionally, a linear regression is performed for one mesh size and the slope is indicated. The measurements were made on the Neon machine and the maximum time of all ranks is shown.

Figure 4.31 shows that the fraction of time used by the mapping decreases with higher number of processors and, thus, that the interpolation process does not pose an upscaling bottle neck as processor counts increase.

Figures 4.32, 4.33 and 4.36 show upscaling results from the Neon[2] machine on up to 64 processes for participant B and a fixed number of 2 processes for participant A. In this case, no inter-node communication took place. Results are shown for a strong scaling setup, i.e., the global problem size n remains constant as the number of processors grows, as well as for weak scaling. The weak scaling setup uses a fixed mesh size per rank. This corresponds to a fixed dimension of the matrix per rank.

Looking at the absolute times, the global basis functions, i.e., volume splines and thin-plate splines need tremendously more time compared to Gaussian basis functions. Thin-plate splines with 50^2 mesh elements are roughly on par with Gaussian ($m = 4$) and 600^2 mesh elements, though the mesh size is smaller by a factor of $1/144$. This, of course, comes at no surprise, as the global basis functions produce densely populated system matrices.

[2] 72 × Intel Xeon E7-8800 @ 2.3 GHz

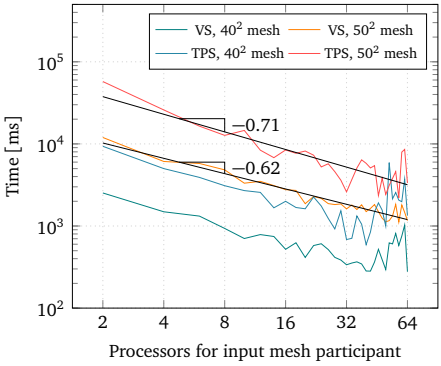


Figure 4.33: Strong upscaling on different mesh sizes with volume splines and thin-plate splines basis functions on up to 64 processors. The time spent in the `computeMapping` event is shown, i.e., the entire offline phase of the mapping. Additionally a linear regression is performed for one mesh size and the slope is indicated. The measurements were made on the Neon machine and the maximum time of all ranks is shown.

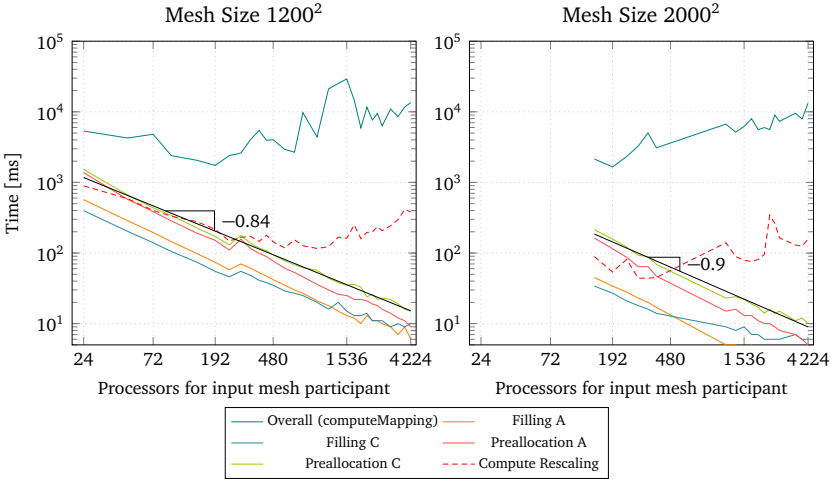


Figure 4.34: Strong upscaling for mesh sizes 1200^2 and 2000^2 with Gaussian ($m = 6$) basis functions on up to 4224 processors on the HazelHen HPC system. Additionally to the `computeMapping` event, timings for the different phases of the algorithm are shown. The slope for the `Preallocation C` event is shown.

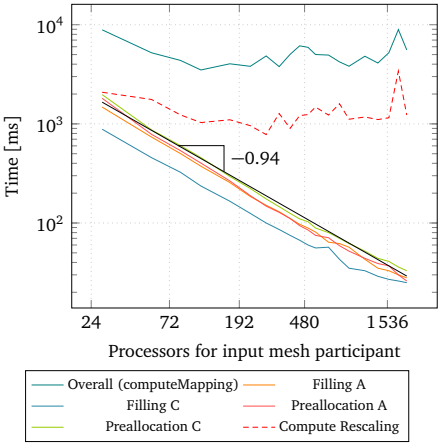


Figure 4.35: Strong upscaling for a mesh size of 1000^2 with Gaussian ($m = 6$) basis functions to up to 2016 processors on the SuperMUC HPC system. Additionally to the `computeMapping` event, timings for the different phases of the algorithm are shown. The slope for the `Preallocation C` event is shown.

For the number of non-zero elements in C, an upper bound can be given from the support radius and mesh size n as $0.5 \cdot \pi\, m^2 \cdot n^2$. For global basis functions, on the other hand, the number of non-zero elements is approximately $0.5 \cdot n^4$. The approximative factor of 0.5 stems from the symmetry of C. Volume splines consistently perform better than thin-plate splines. This can be attributed to the computation of the logarithm in the thin-plate spline function, needing considerably more CPU cycles than the simple volume-splines function which only involves computing the absolute value of the argument. Due to the low performance, especially as meshes grow larger, the global basis functions will not be considered for larger upscaling runs.

For Gaussians, the shape parameter has only little influence on the upscaling slope, ranging from −0.83 to at worst −0.69 for a mesh size of $n = 200$ and $m = 4$. In general, for smaller runs and with Gaussian basis function, the RBF implementation shows a satisfactory upscaling behavior.

Larger runs are performed on the HazelHen and SuperMUC HPC systems, see Figs. 4.34 and 4.35. The runs still exhibit some problems when upscaled to higher number of cores. While the core parts (preallocation & filling), which are the computationally most expensive parts of the algorithm when executed on a single node show an excellent scaling with an

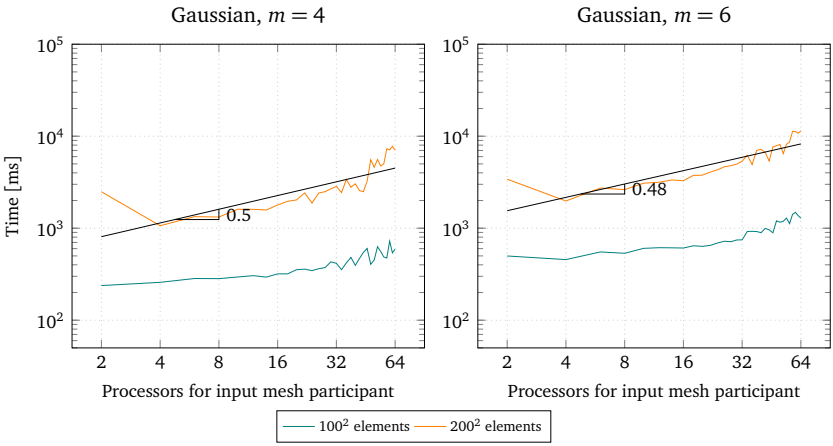


Figure 4.36: Weak upscaling on different mesh sizes with Gaussian basis functions to up to 64 processors. The time spent in the `computeMapping` event is shown, i.e., in the offline phase of the mapping. The mesh size indicated in the legend is the base mesh size for one rank. Additionally a linear regression is performed on one run and the slope is indicated. The measurements were made on the Neon machine and the maximum time of all ranks is shown.

exemplary slope of −0.84 for the preallocation part, the overall scaling leaves room for improvement. Closer scrutinization of the different parts of `computeMapping` reveals three parts that become significant at higher numbers of processors and possibly also due to increased inter-node communication. Visible in Figs. 4.34 and 4.35 is that the time for the computation of the rescaling interpolant stagnates. Not shown here is the significant time that is taken by setup of the PETSc distributed matrices and initialization of the `AO` (application ordering) object as part of the computation. Both operations are collective on the respective communicator and involve multiple collective communication operations, e.g., `AOCreateMapping` dispatches three calls to `MPI_Allgather` which initiates an all-to-all communication.

4.9.2 Online-Phase

Certain parts of the algorithm are executed for each interpolation step, i.e., whenever the right-hand side f_ξ of Eq. (2.25) changes. The phase mainly consists of copying data between PETSc and preCICE data structures and solving the linear system $C\,\boldsymbol{\lambda} = f_\xi$. Note that parts of the solving procedure, such as factorization for the preconditioners are already accomplished during the offline-phase by the computation of the rescaling interpolant, which solves the linear system for a different right-hand side. The plots henceforth distinguish two phases, the overall execution time of the function `mapData` and the time to solve the linear system. The latter being part of the overall timings of `mapData`.

Figures 4.37 to 4.39 show strong scaling of the overall mapping time for each interpolation step and the time consumed for solving the linear system. Compared to the offline-phase which setups the interpolation, the online phase needs less time. For Gaussians, copying between data structures of preCICE and PETSc dominates the time for a higher number of cores, the share used for solving the linear system drops from more than 60% to 14% of the total time of `mapData` as the number of processors increases to 64. The measurements for a larger number of processors on the HPC systems confirm that observation. With thin-plate splines and volume splines, on the other hand, solving the densely populated linear system dominates over copying data structures.

The scaling of the online phase is much less monotonic than the results of the offline phase, with the SuperMUC system being a laudable exception. At higher number of processors, the workload per processor diminishes, e.g., at 4224 processors, the mesh size is only around 15×15 vertices per rank. Tasks that become more expensive with higher number of processors such as communication and setup of parallel data structure likely dominate the timings in this case.

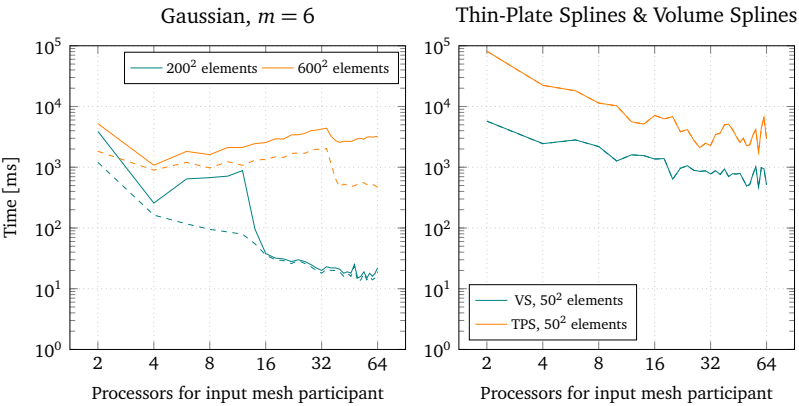


Figure 4.37: Strong upscaling of the online-phase at Neon machine. The overall needed time is plotted with solid lines. The dashed lines indicate the time needed for solving the linear system. For thin-plate splines and volume-splines the dashed and solid lines are practically identical.

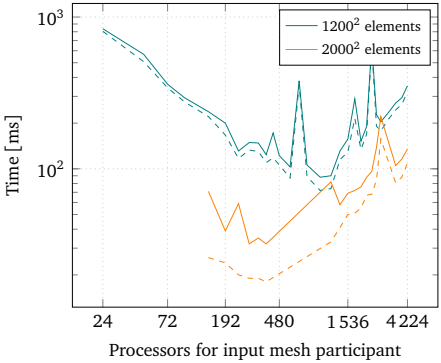


Figure 4.38: Strong upscaling of the online phase with a mesh size of 1200^2 and 2000^2 with Gaussian ($m = 6$) basis functions to up to 4224 processors on the HazelHen HPC system. The overall needed time is plotted with solid lines. The dashed lines indicate the time needed for solving the linear system.

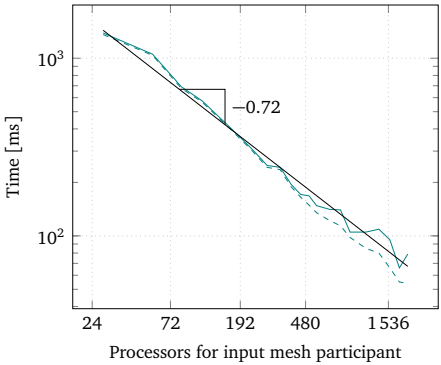


Figure 4.39: Strong upscaling of the online phase with a mesh size of 1000^2 with Gaussian ($m = 6$) basis functions to up to 2016 processors on the SuperMUC HPC system. The overall time of `mapData` is plotted with solid lines. The dashed lines indicate the time needed for solving the linear system.

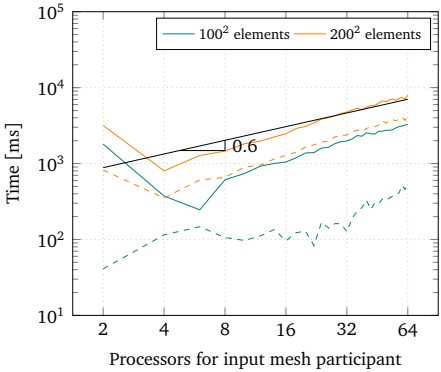


Figure 4.40: Weak upscaling of the online-phase at Neon machine, based on mesh sizes of 100^2 and 200^2 on 2 ranks and Gaussian ($m = 6$) basis functions. The overall time of `mapData` is plotted with solid lines. The dashed lines indicate the time needed for solving the linear system. For thin-plate splines and volume-splines, the dashed and solid lines are practically identical.

Figure 4.40 shows a weak scaling to up to 64 processors. In a weak scaling scenario, the optimal slope is the constant horizontal line. The moderate slope of 0.6 can still be considered adequate weak scaling, whereas measurements to higher number of processors are required to make definite conclusions.

The results of the presented upscaling runs exhibit a quite diverse and sometimes inconclusive behavior. This is the result of various effects that determine the measured results. First, the runtime of the purely local portions of the algorithms, preallocation and filling do not require any communication between processors. On the other hand, computation of the rescaling interpolant as well as solving the actual interpolation system involve heavy communication across ranks. While nodes are assigned to a compute job exclusively, the network must be considered a shared medium with other users and, therefore, adding fluctuations to the measurements. The entire process (preallocation, filling as well as solving) again depends on the mesh decomposition. The decomposition of the two-dimensional mesh is performed by ASTE, using the *uniform* algorithm, see Sec. 6.2.3. It strives to achieve a quadratic partitioning, i.e., to have an equal number of partitions in x and y direction as far as possible. Some, most notably prime, number of processors generate decompositions, that only have one partition in one direction, influencing the interpolation algorithm.

Still, the data shown in Fig. 4.38 is lacking a valid explanation. As the measurements are reproducible, possible explanation approaches can be found in the mesh decomposition as outlined above and in the various tuning parameters set by the vendor for the MPI and PETSc libraries.

Overall, the upscaling runs yield satisfactory results, with the principal parts of the algorithm exhibit a good scaling behavior. Potential impediments for scaling are the setup of the PETSc matrices and the application ordering objects. In the future, it might be possible to replace the latter by a changed vertex enumeration scheme in preCICE.

4.10 The Cylinder Flap Test Case

In this section, I will finally examine a “real” fluid-structure interaction case with actual solvers involved. While my focus is still on the interpolation, I observe the effects of different interpolations on the outcome of the simulation and not on isolated properties, such as condition number or accuracy, as it is the case in the previous sections.

This two-dimensional case consists of a laminar incompressible channel flow. In the channel, attached to a rigid cylinder, an elastic flap produces self-induced oscillations and vortex shedding downstream. See Fig. 4.41 for a visualization. This case resembles the

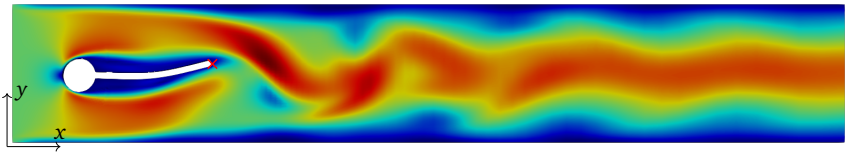


Figure 4.41: The cylinder flap test case with coloring according to the absolute velocity magnitude.

widely used Turek test case [103]. The geometry is intentionally slightly asymmetric to ensure a consistent onset of the flap oscillations independently of the precision of the computation. A difference from the reference test case is the inlet profile. Whereas Turek uses a parabolic profile, I use a uniform, time-dependent inlet velocity profile. The boundary condition ramps up the velocity, such that the final velocity of $\boldsymbol{u} = (2, 0, 0)^T$ is reached at $t = 2$. This helps to stabilize the simulation. With the given inlet velocity, the resulting Reynolds number is $\mathrm{Re} = 200$, based on the cylinder diameter as the characteristic length measure. The case setup corresponds to the FSI3 setup from [103]. The parameter settings are summarized in Tab. 4.7.

The channel flow is simulated using the OpenFOAM *pimpleFoam* solver and the CALCULIX finite-element solver acts as a structure solver for the elastic flap.

Two setups are compared, differing in the size of the mesh. The coarse case consists of 6306 cells in the fluid domain and 280 elements the structural domain. From that, the fine case is derived by doubling the resolution in each direction, resulting in 26 772 cells and 1120 elements, respectively. The time step size in both cases is $\mathrm{dt} = 0.001$.

This test case is used to investigate the influence of different interpolation methods and mesh sizes on the dynamic behavior of the flap. For that, the displacement of the elastic flap in y-direction is studied, the point probe location is marked in Fig. 4.41 by a ×.

The periodic displacements of the flap for different basis functions are plotted in Fig. 4.42. The used basis function has little influence on the amplitude and frequency on both the coarse and fine grid. All interpolations on the coarse mesh exhibit a dominant frequency of $4.5\,\mathrm{s}^{-1}$ and an associated amplitude of $0.01155 \pm 2.06 \cdot 10^{-5}$. The spectrum shown in Fig. 4.43 does not show significant differences between different interpolations on the coarse mesh.

The finer mesh resolution has a much stronger effect than the interpolation parameters. The dominant frequency on the finer mesh is $4.1\,\mathrm{s}^{-1}$ with an associated amplitude of $0.0097 \pm 2 \cdot 10^{-8}$. The results together with the reference results are shown in Tab. 4.8.

Parameter	Value	Unit
Fluid density ρ_f	1000	kg/m^3
Fluid kinematic viscosity ν_f	0.001	m^2/s
Inlet velocity u	2	m/s
Structure density ρ_s	1000	kg/m^3
Poisson's ratio ν_s	0.4	–
Shear modulus μ_s	$2 \cdot 10^6$	$kg/m\,s^2$
Reynolds number Re	200	–

Table 4.7: Physical parameter settings for the cylinder flap test case

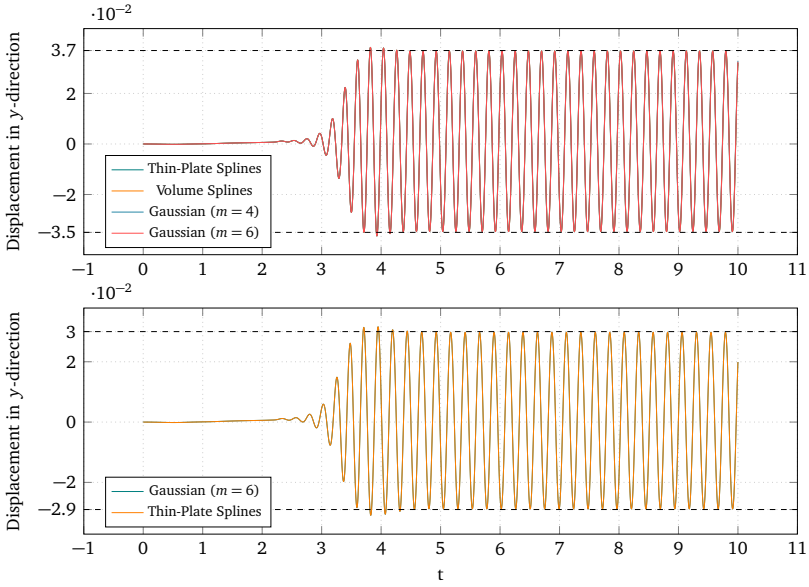


Figure 4.42: Displacement of the flap in y-direction of the cylinder flap test case on the coarse mesh (top) and fine mesh (bottom) with the maximum and minimum amplitudes marked.

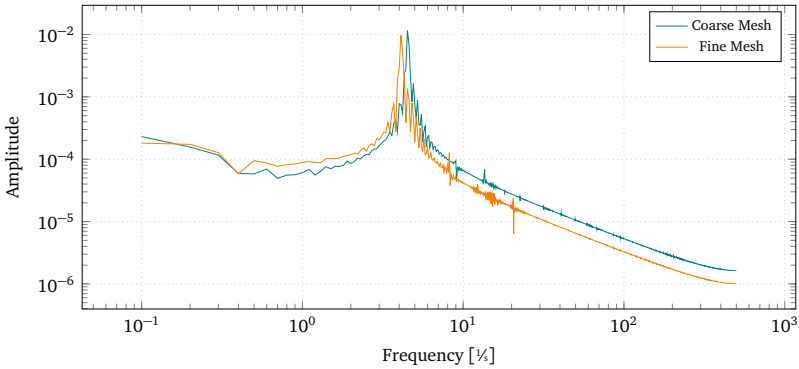


Figure 4.43: Spectrum of flap movement in y-direction of the cylinder flap test case for Gaussian basis functions ($m = 6$) on different meshes, computed by using Fourier analysis.

Basis Function	Mesh	Displacement $[\cdot 10^{-3}]$	Frequency $[s^{-1}]$
Volume Splines	coarse	1.064 ± 35.727	4.5
Thin-Plate Splines	coarse	1.063 ± 35.728	4.5
Gaussian ($m = 4$)	coarse	1.062 ± 35.723	4.5
Gaussian ($m = 4$)	coarse	1.062 ± 35.727	4.5
Thin-Plate Splines	fine	0.502 ± 29.279	4.1
Gaussian ($m = 6$)	fine	0.500 ± 29.279	4.1
Reference from [103]	–	1.48 ± 34.38	5.3

Table 4.8: Results of the cylinder flap test case. The displacement in y-direction is shown as mean ± amplitude.

Although, the frequency of $4.1\,s^{-1}$ on the fine mesh agrees with the vortex shedding frequency of a cylinder which is, according to Strouhal [98],

$$\mathrm{St} = \frac{f \cdot d}{u} \approx 0.21 \quad \Rightarrow \quad f = 4.2\,s^{-1}, \tag{4.21}$$

the accordance with the results from Turek and Hron leave plenty of room for improvement. The reference frequency is given as $5.3\,s^{-1}$. The results on the coarse grid generally show better accordance with the reference results. A possible reason for the underestimated frequency are linear elements that are used in modelling the elastic flap and likely result in an exaggerated stiffness. Using higher-order elements might result in more accurate results here.

4.11 Conclusions

This chapter provides a broad overview of radial-basis function interpolation under different parametrizations.

Naturally, the most important parameter of an RBF interpolation is the type of basis function. In this section, I evaluated Gaussians, compact polynomial C0 splines, thin-plate splines, volume splines and multiquadrics functions. Publications looking more at the mathematical side of the algorithm [45, 90, 91, 110] propose a number of additional basis functions, such as Bessel, Wendland functions or Sobolev splines. More engineering oriented papers [6, 10, 101] rather focus on thin-plate splines and Gaussians for compact basis functions. For compact basis functions, the support radius plays a decisive role for stability and accuracy of the interpolation. To facilitate comparability between different shape parameters, I use an alternative formulation in terms of the number of vertices m included in the basis functions support in each direction, in contrast to the shape parameter ϵ, which has different meanings depending on the basis function.

Multiquadrics functions are quickly ruled out as a suitable basis function choice due to their inability to reach a stable solution. Without surprise, the remaining global basis functions, volume and thin-plate splines, give the best results in terms of accuracy among all functions tested. With the general exception of the discontinuous jump test function, both basis function show consistently 2^{nd}-order convergence in the input mesh width h for volume splines and 3^{rd}-order convergence for thin-plate splines. The convergence rates are independent of whether a polynomial is added to the interpolation process or not.

The compact basis functions, namely Gaussians and compact polynomials perform worse

than the global functions. This is a well-known fact and goes so far as "there is an ongoing discussion [in the approximation theory community] about usefulness of radial basis functions of compact support because of their inferior convergence properties [...]." [19]. I would not go so far as to dispute the usefulness of compact basis functions, but their numerical properties tend to be much more intricate. The stability greatly depends on the support radius chosen. For Gaussians, I explore a range of support radii $m = 4 \dots 16$. Only $m = 8$ to $m = 12$ prove to useful. Smaller support radii achieve stable interpolation, though with inferior accuracy while support radii too large cause the interpolation to become unstable and with that also inaccurate. The effect of error saturation prove to be a lower bound for the accuracy in the range of 10^{-9} for the relative L_∞-error. The stability range of support radii for compact polynomials is much larger, albeit the accuracy remains inferior to Gaussians. Like Gaussians, compact polynomials are also subject to error saturation. For compact basis functions with small support radii, the rescaled interpolant provides an effective measure to improve the accuracy and adds only little additional computational cost to the algorithm.

Augmenting the RBF interpolant by a linear polynomial has little influence on the convergence rates, but consistently improves the absolute accuracy for all basis functions, depending on the y-offset and linear trend of the input data. The integrated polynomial worsens the condition number of C. Global basis functions experienced a larger condition number, but no increase in the condition number with decreasing input mesh width. For compact basis function, the condition number does increase with decreasing input mesh width. The separated polynomial has no effect on the condition number at all, but improves accuracy in a similar way as the integrated polynomial.

Non-uniform meshes are a special challenge for compact basis functions, as the optimal shape parameter is not constant over the mesh. Adaptive shape parameters, depending on the local mesh width can improve this situation, but come with a number of downsides impairing the numerical properties, such as loosing the symmetry in C. Furthermore, knowledge of the local mesh density is required, which is usually not provided by the solver.

The basis transformation of C performed by the RBF-QR algorithm provides promising first results, but also shows weaknesses for certain situations, such as interpolation near boundaries. Furthermore, for each basis function an analytic expansion needs to be found first.

Evaluating conservative interpolation requires a new approach in measuring the error. For that, I propose a new *rescaled error* measure, which accurately reflects the local interpolation error.

Results show that interpolation by RBF is powerful and works well on uniformly scattered data. On uniform meshes, the width m of the basis functions is an effective parameter to control the accuracy-condition trade-off and is applicable to meshes of different sizes. For non-uniform meshes, the choice of a suitable support radius remains a challenge. Nevertheless, the methods presented here have improved interpolation in that case as well. To correctly use RBF interpolation, a sizable amount of expert knowledge regarding the choice of parameters is required from the user. Future work should look into automatisms for optimal parameter selection.

Mainly thanks to the re-partitioning scheme and using the highly efficient PETSc numerical library, the RBF implementation achieve good parallelism and a satisfactory upscaling behavior. The offline-phase dominates the timings and due to the constant linear system during the simulation the online-phase is no bottleneck for upscaling multi-physics simulations. Still, the upscaling also revealed problems which are, however, mostly out of scope for preCICE and need to be addressed by the HPC vendors.

As a last and final plot in this chapter, I would like to show Fig. 4.44. It gives an overview in terms of condition number and RMSE accuracy as the support radius grows larger over a selection of the methods previously presented. While this in no way can replace thorough study of parameters, it shows the general effectiveness of the improvements presented in this chapter.

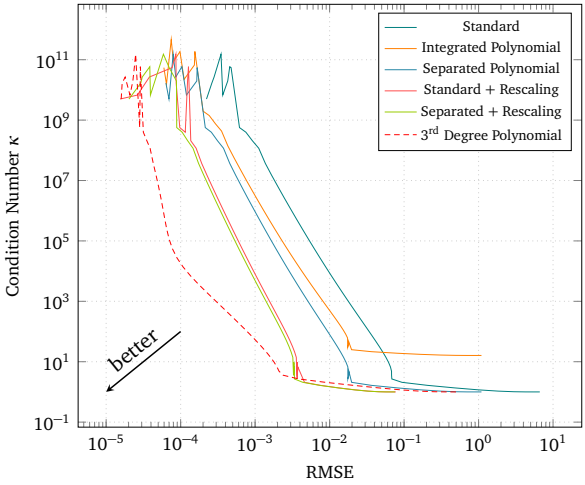


Figure 4.44: Comparison of different methods on a mesh of size 192 for the high frequency test function. Basis functions used are Gaussians with support radius $m = 1 \ldots 16$. The RBF implementations compared are: *Standard:* bare RBF interpolation without polynomial, *Separated/Integrated Polynomial:* using a linear polynomial in either the separated or integrated formulation, *Standard + Rescaling* and *Separated + Rescaling:* is either the standard RBF implementation without polynomial or the linear, separated polynomial formulation with additional rescaling, *3rd Degree Polynomial:* uses a separated polynomial of third (cubic) degree.

5 Efficient Communication on HPC Machines

> You don't have to be responsible for the world that you're in.
>
> John von Neumann

Efficient communication is of paramount importance for preCICE. The partitioned type of coupling requires data exchange at each coupling iteration, which can be multiple times per timestep.

In this chapter, I will shortly describe the fundamental communication channels as well as suitable communication protocols. A method to establish peer-to-peer communication networks on high-performance computers is presented and tested for scaling. Furthermore, a technique to extend the amount of asynchronicity between participants is presented.

5.1 Communication in preCICE

As a coupling tool, efficient communication inside and between participants is one of preCICE's prime objectives.

The basic connection topology is created by the process described in Sec. 1.3.2. The selected mapping defines which connections are to be established by marking vertices as required to perform the mapping, see Sec. 3.4.2. In this chapter, I will focus on the lower-level parts of the communication.

preCICE uses three different communication channels for steering the simulation, exchange of coupling data between participants, and intra-participant communication. Figure 5.1 shows these three different communication channels. Accordingly, we require the following types of communication:

Master Communication provides a channel between the masters, i.e., rank zero of each participant. Master communication is mostly used for exchanging meshes at the initialization of the simulation, coordination of the participants during setup and

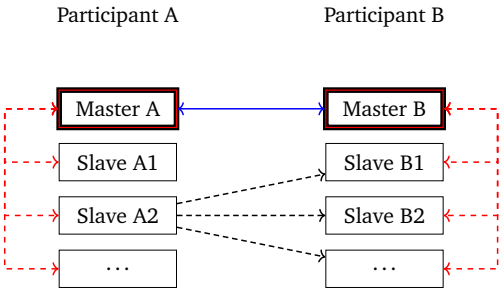


Figure 5.1: Basic types of communication schemes used in preCICE: Master (___), Master-Slave (_ _ _) and M2N communication (_ _ _)

tear-down as well as steering of the overall simulation, e.g., deciding on convergence of coupling iterations.

M2N Communication is established between slaves of both participants. It is used to exchange coupling data between participants during the simulation run.

MasterSlave Communication is used for communication between the master and slaves of a participant. Meshes received from the other participant's master are distributed to the slaves via masterslave communication. It is also involved in collective intra-participant operations performed by the quasi-Newton algorithms.

From these three communication channels, two different communication domains can be identified. Communication within one participant and communication between participants. While similar in the used techniques and protocols, fundamental differences how the communication is established exist.

5.2 Fundamental Communication Protocols

Before comparing possible candidates for communication protocols in the field of partitioned multi-physics simulations, I will provide a set of diverse criteria that can be used to compare different protocols.

Availability The communication protocol has to be available on all supported systems, ranging from personal computers to large high-performance machines. This also

includes the actual implementation status available at a particular system. A complex API like MPI may have the core parts available, while lacking in other areas.

Bandwidth mostly depends on the underlying networking hardware. As communication protocols are usually optimized towards high bandwidth, the criterion can be considered as granted.

Latency Data can be exchanged multiple times per time step. Algorithms, such as linear system solvers do a high-number of small sized transfers across the nodes. This makes low latency communication an important factor [1, page 20].

Communication Scope While a lot of IPC protocols, such as D-Bus or COM fulfill the requirements above, their scope is usually on the local system only. Large distributed simulations require communication across node boundaries.

Reliability Applications built upon a certain protocol must be able to rely on certain features. This includes guarantees of message ordering, i.e., messages are not able to overtake each other, and reliable delivery. This frees the application from the burden of implementing its own transmission control layer.

Asynchronicity While asynchronous send or receive operations usually are considered a client-side concern, some protocols offer special support for them.

Topology Most protocols natively only support one-to-one connections, while others also provide collective operations, such as broadcasting or even reduction operations. Especially in scientific computing, efficient implementation of collective operations can provide gains in performance.

The **Message Passing Interface (MPI)** is a standard for a message oriented communication protocol geared towards high-performance computing. The standard [48] defines the methods and their interfaces as well as semantics. Based on the standard, different implementations exist, most notable, OpenMPI [49] and MPICH [54]. HPC vendors usually tailor their hardware for efficient support of MPI messages. An example is the Aries network of Cray XC40 machines [1].

The Open Systems Interconnection model (OSI) [63] defines a layered model to characterize network protocols. The following protocols are located at the network layer 3, the transport layer 4 and the session layer 5. The *network layer* is responsible for addressing and routing of data packets though the network. The higher *transport layer* controls reliability of a communication link by addressing flow-control, segmentation and desegmentation

(splitting of packets that exceed the maximum transfer size) as well as error control. On the *session layer* 5, the establishment, management and termination of connection takes place. From the operating system the connection management is abstracted by means of *sockets*.

TCP/IP and UDP/IP are a family of communication protocols which are ubiquitous on modern computing systems. The lower-level *Internet Protocol* (IP) [60] or its successor, the IPv6 [30] protocol is responsible for the addressing of nodes in the communication network. It corresponds to the network layer 3 in the OSI model [63]. Both TCP and UDP are located on the transport layer 4 of the OSI model.

The *User Datagram Protocol* (UDP) [81] is a connection-less, state-less protocol based on IP for data transfer in packet-switched communication networks. Its lack of a transmission control layer, i.e., a mechanism that detects failed transmissions of packets and repeats the transmissions, makes it a bad choice regarding reliability.

This re-transmission mechanism is available in the *Transmission Control Protocol* (TCP) [61], which is a state-full, connection oriented communication protocol, also based on IP. In the following, for easier readability, I will refer only to TCP when talking about TCP/IP.

Given a **distributed file system**, shared among all participating nodes, files can be used for communication. While this method shares the universality of TCP sockets, it becomes prohibitively slow and therefore lacks important features, such as low latency and high bandwidth.

When compared to MPI, TCP is still considered a low-level protocol. It is programmed by using *sockets* provided by the operating system. These sockets accept a stream of bits. In contrast to this, MPI accepts messages that have a certain data type. TCP also lacks collective operations provided by MPI such as broadcasting or gather/scatter. MPI also provides support for asynchronous sending and receive operations, which is not available with TCP per se. With these semantics, MPI can be placed at the presentation layer 6 of the OSI model. The MPI implementation translates between application data formats (e.g. an array of doubles with associated size) and network formats (the resulting stream of bytes).

5.2.1 Protocol Semantics & Abstraction in preCICE

MPI provides powerful communication primitives, geared towards patterns that typically arise in scientific computing. This includes one-to-one communication methods in synchronous and asynchronous variants. Similar one-to-one communication semantics are also provided by TCP. A communication handle, which is a *communicator* for MPI can be found in the form of a *socket* for TCP. Here, first important semantic differences arise. While an MPI communicator can hold an arbitrary number of connections or ranks, a TCP socket

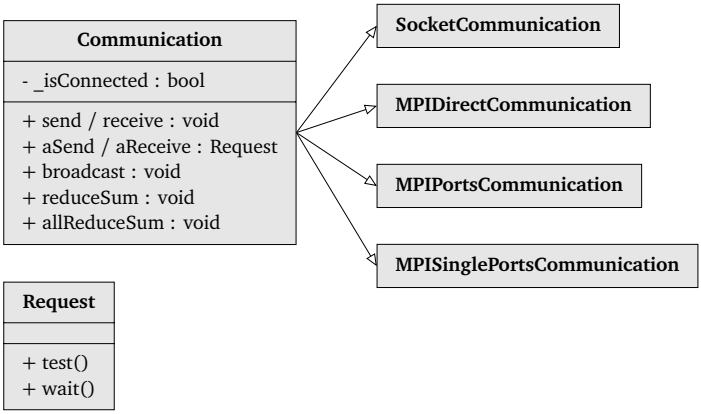


Figure 5.2: Communication related classes in preCICE.

can only hold one connection, identified by an address/port tuple for each side of the connection.

MPI knows a number of collective operations such as broadcast (one-to-many), gather (many-to-one) or more complex ones, like reduction or distributed I/O. In addition to the communication operations, MPI also offers operations to produce new communicators from sets of ranks or from existing communicators as well as synchronization operations such as barriers. TCP, however, not aware of multiple peers, is devoid of these collective operations.

For that reason, an abstraction layer is needed in order to provide a uniform set of methods for communication. This abstraction layer is designed with no intention of emulating every MPI operation using TCP primitives, but a restricted set of required and relatively easy to implement methods. The low-level socket handling is implemented using the *Boost Asio*[1] (which stands for asynchronous IO) library.

Synchronous send/receive operations are the most straightforward and the most similar in MPI and TCP. For sending arrays, TCP requires two send-receive operations to transfer the size first, for MPI, the `Get_count` function can be used instead.

Asynchronous send/receive is implemented with MPI using `Irecv` and `Isend`, respectively. Both return a request handler, which is encapsulated by a `Request` object, see Fig. 5.2. For sockets, a callback mechanism is used on the returned `Request` object.

[1] `https://www.boost.org/doc/libs/1_69_0/doc/html/boost_asio.html`

The `test` or `wait` methods can then be used to check for completion of the request. Additionally, for sockets to ensure correct ordering of multiple asynchronous requests, a queuing technique is employed, see Sec. 5.4.2. For MPI, the correct order of messages is guaranteed by the standard.

Broadcast and reduction are collective operations over multiple ranks, which are not supported for sockets. Therefore, they are replicated by a looped send over all ranks. Reduction operations are performed by collecting all input data at one particular rank, performing the reduction and finally broadcast the result. Currently, only the *reduce-sum* operation is implemented.

From the description above, it is obvious that for the given case, MPI is clearly the preferred communication protocol. Its support for collective operation not only eases implementation, but may lead to increased performance by using optimized algorithms. Note that, however, no such performance guarantees are given by the MPI standard. The correct ordering of messages, on the other hand, is guaranteed by the standard and constitutes an important factor when compared to TCP.

The actual intrinsics of the different protocols are abstracted from the higher levels of preCICE.

5.3 Establishing Peer-to-Peer Communication Networks

In this section, I concentrate on the technical implementation of the communication, in particular establishing communication channels between ranks of different participants. The algorithmic side, i.e., computing the communication topology is described in Sec. 1.3.2.

As outlined above, we use TCP and MPI as the fundamental communication protocols. TCP defines it communication domain as all IP addresses which are routable. MPI, on the other hand, creates a communication domain for each application it starts, usually with the `mpiexec` binary. Coupled simulations are usually started by running each solver inside a distinct communication domain. To connect these domains, the *ports* part of the MPI standard is used.

5.3.1 Implementation Status of TCP and MPI

TCP implementations at the target operating systems, which is Linux for HPC applications and furthermore Microsoft Windows and Apple MacOS X can be considered mature and feature complete.

For MPI, the picture is less clear. MPI ports are a part of the MPI Dynamic Process Management (DPM), which appeared in version 2 of the standard, released in June 1997. Though being around for a long time, the support of this part of the standard is still incomplete in major implementations. Reason is, as I have been told by operators, that DPM is a rather rarely requested feature. Most parallel application purely rely on the point-to-point and collection operations parts of the standard.

Cray MPI is built on MPICH2 distribution from Argonne [1, page 20]. A white paper [77] details the implementation status of the dynamic process management. On Cray machines, DPM can be supported through two different techniques, either *pdomains* or *Dynamic RDMA Credentials* (DRC). The DRC require the Cray Linux Environment at version CLE 6.0 UP06. The HLRS HazelHen machine is at UP05 as of August 2018. I therefore use *pdomains* to achieve DPM capabilities.

Tests on SuperMUC were done with the software stack on phase 2 as of August 23, 2018 using modules `mpi.intel/2017_gcc`, `gcc/6`. The Intel MPI implementations we use at SuperMUC supports MPI ports, albeit only for communication not involving multiple islands of the computer. This restriction is not specific to MPI Ports, but generally applies when using the Intel MPI implementation.

With MPICH, I experienced the least problems. The implementation fully supports the MPI ports features I tested for.

On its homepage, OpenMPI claims full MPI-3.1 compatibility. However, to connect two OpenMPI applications, a central instance of the `ompi-server` is required. This server can be used to exchange the connection information (see Sec. 5.3.2) by means of `Publish` and `Lookup` and acts as a key-value store. While this is certainly covered by the standards, it is an additional pitfall as the external server needs to be launched by the user prior to starting the application. Furthermore, it requires an additional code path, as exchange of connection information via a third party, e.g., file system is not possible.

The information given above makes it clear, that, while MPI is clearly preferable, an alternative, fallback protocol must be available. This protocol is TCP.

5.3.2 Exchange of Initial Connection Information

In order to establish a connection between two processes at different solvers, a token that identifies the acceptor of the communication link needs to be exchanged. For TCP/IP connections, that token is the IP-address and port of the listening socket, for MPI Ports connections, it is the information acquired from `MPI_Open_port`. Exchanging that token requires a common communication domain to exist a priori. For most applications, the only communication domain which can be taken for granted is a network file system, i.e., a file system which is shared among all nodes that participate in the simulation.

The number of connections that are to be established depends on the decomposition of the geometry among the ranks and the mapping method chosen. Given n_A ranks at the acceptor participant, n_B ranks at the requester and m meshes to be transferred, the number of connections is $n_A \cdot n_B \cdot m$ when a mapping method that uses all vertices, such as RBF with a global basis function is used. The number of meshes are a factor due to the current implementation in preCICE which does not reuse existing connections, but establishes an own set of connections for each mesh to be mapped.

Using a global mapping method is discouraged for large simulations, but still the number of connections can be significant. For each connection, one token must be transferred to the requesting rank.

Most parallel file system experience performance degradation, when too many files or directories are created within one directory. What exactly "too many" is, varies greatly among file systems and individual settings of the system's operator. The LRZ, operator of the SuperMUC HPC system considers as little as 1000s of files per directory as too many for the IBM General Parallel File System (GPFS) [84].

The original scheme implemented in preCICE was to create files, named after `.A-B-4.address`, which holds the connection information needed to connect to rank 4 on participant B from participant A. For each mesh, a separate set of connections is established, as the connection patterns are potentially different for each.

For runs on a large number of ranks, this quickly turns out to be an hindrance to efficient upscaling. It was subsequently augmented by a workaround that introduces an additional level of directories named `.B-MeshName-4.address/`, for connection info files to rank 4 on participant B for mesh MeshName. Connection info files holding information on connections to participant B for that particular mesh are henceforth placed in that directory. Thus, we create one folder per mesh containing one per file per rank of participant B. The total number of files remains constant, but the files are split up across different directories

for each mesh and receiving participant. This workaround proved to be effective in moving the limit to a higher number of connections which, however, will inevitably be reached soon. For that reason, a more flexible and definitive scheme is proposed, implemented and tested here.

The proposed scheme composes a string from the name of the accepting and the requesting participant and the accepting rank. The resulting string is then hashed using the SHA-1 [38] algorithm. The resulting 160 bit number can be represented as a string of 40 hexadecimal digits. The first two digits are used to define the directory name, the remaining 38 characters are used as the name of the connection info file stored in the respective directory.

As an example, the connection information from the request above, i.e., from A to rank 4 of B, is placed into a file `precice-run/62/2a190c33ecf761d0ea546bc48696c1270f3c16`.

While, at first glance, this might appear needlessly complex, the scheme features a number of unique advantages. Due to the avalanche effect of hashing functions, similar input data are mapped uniformly to the entire image of the hash function. This ensures that the created files are spread evenly among the directories. The number n of digits cut from the hash to form the directory determines the maximum number of directories as 16^n. This way, the scheme can be tuned to the characteristics of the underlying file system. By choosing other number representations, the number of directories can be set in steps other than powers of 16. Alternatively, to control the number of entries per directory, the scheme can be extended to generate more than one directory level from the hash. Furthermore, other information than the participants names and the rank of the recipient can easily be incorporated into the hashed string, making it future proof to modifications of the connection scheme.

It is well known, that the SHA-1 hashing function can be considered broken and should not be used anymore. This recommendation, however, is only relevant for cryptographic purposes which is not the case here.

In contrast to the node's CPU, the file system is a shared resource among all users and as such afflicted with a high variance between measurements. For that reason, Fig. 5.3 shows a statistical breakdown of 5 runs on the SuperMUC and HazelHen HPC systems. The figure compares the old scheme of writing connection info files with the new one, using the hashing naming scheme. As the mapping is responsible for the connection pattern between the participants (whom connects to whom?), different mappings results in different number of connections to be established. For the setup used on SuperMUC, nearest-neighbor

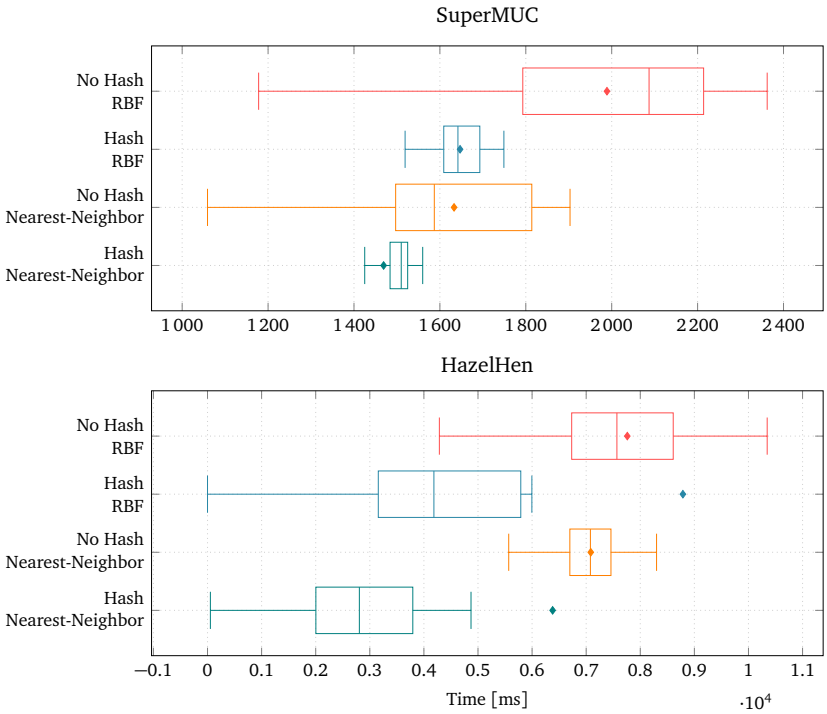


Figure 5.3: Performance comparison between connection info files naming schemes with and without hashing for two different mappings. 5 measurements on for each configuration are made, using 280 processors per participant on SuperMUC and 600 processors on HazelHen. The box spans the 0.25 to 0.75 quantile of the data (interquartile range), the whiskers indicate the smallest respectively largest datum that is still contained in 1.5 times the interquartile range. Further the median is marked with a vertical line, the mean with a rhombus.

mapping results in an average of 1.19 connections per rank, while the RBF mapping with a Gaussian basis function and $m = 6$ results in 8.2 connections per rank. The connection numbers for the mesh setup used at HazelHen are 1.2 for nearest-neighbor and 68 for RBF interpolation.

The new variant, using the hashed filenames and directories consistently reduces the time needed for writing the connection information to the file system. It also reduces the variance, as can be seen by the smaller interquartile range visualized as boxes in the plot.

5.3.3 Creating the Communication Channels

preCICE implements four variants basic communication variants.

SocketCommunication is based on TCP. It is mainly used for master and m2n communication. In cases where MPI can not be used for master-slave communication, which can be the case with proprietary solvers using a closed MPI variant, it can also be used for intra-participant communication.

MPIDirectCommunication connects two disjoint groups of MPI ranks. These groups, however, must be represented in a common communicator and are usually the result of a communicator split. This communication can only be used for master-slave communication, i.e., intra-participant communication.

MPIPortsCommunication and *MPISinglePortsCommunication* are two variants that both use the MPI ports functionality to create a communicator from two sets of independently launched MPI processes. The first variant creates a communicator for each pair of connected ranks, resulting in a large number of communicators. The latter variant, on the other hand, uses one communicator that encompasses all ranks from the participants, thus resulting in just one communicator per participant. These two variants are used for the m2n inter-participant communication. The performance is studied in greater detail below.

The actual connection establishment works alike in all variants and results in one or more communicators or sockets, respectively. A single rank from the other participant is always addressed by its participant-local rank number. The translation to the actually used communicator and rank is performed by the communication classes.

5.4 Scaling & Compatibility

In this section, I will analyze two important performance aspects of a coupled simulation. First, the phases of establishing the communication network using MPI are benchmarked.

Second, a measure to improve the overall scalability of an coupled simulation by reducing dependency between the participants is presented.

5.4.1 Performance of MPI Ports

In this section, benchmark results for the different connection variants are presented. Compared are the connection schemes using a communicator for each connection and using a single communicator for all connections. These variants are implemented in *MPIPortsCommunication* and *MPISinglePortsCommunication*, respectively.

The MPI performance is tested by using a variable number of connections per rank, based on the total number of processors n, where both participants always run on the same number of processors. On SuperMUC, each rank connects to $0.4\,n$ ranks, on HazelHen, with a higher number of ranks per node, each rank connects to $0.3\,n$ ranks. Thus, this scenario corresponds to a weak scaling. However, it is not directly comparable to a strong or weak scaling of a coupled simulation. Given a basis function that is scaled with the mesh, i.e., $m = \text{const.}$, strong scaling of a coupled simulation means that the connections per rank decrease, weak scaling that the connections per rank remain constant, as the number of processors goes up. The amount of data transferred between the two ranks of each participants is held constant with 1000 rounds of transfer of an array of double values of size 500, 200 and 4000 from participant B to A.

Each measurement is performed five times, the fastest and the slowest run are ignored. From the remaining three, the mean is presented here. The times are taken from rank zero, which is synchronized with all other ranks by a barrier, making the measurements from each rank identical. Note, that the measurements are not directly comparable between SuperMUC and HazelHen, as an identical number of ranks does not correspond to an identical number of nodes.

Three phases are compared in Figs. 5.4 to 5.6. (i) Publishing involves acquiring the connection token from `MPI_Open_port` and writing it to the file system. The single ports variant has a conceptual advantage here, as it only needs to write one file from rank 0. For that reason, the timings of this variant, being almost zero are not shown in Fig. 5.4.

(ii) The connection is established using the methods `MPI_Comm_accept` and `MPI_Comm_connect`. This is an implicit synchronization point between the participants and the lines for A and B in Fig. 5.5 are almost identical. Showing a similar scaling behavior, the single ports variant consistently performs two orders of magnitude better than the old approach.

(iii) The timings for the actual data transfer shows little difference between the variants in Fig. 5.6. The single ports variant for small amounts of data at SuperMUC is an exception

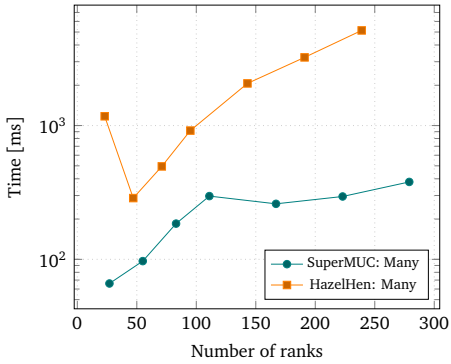


Figure 5.4: Publishing of MPI connection information from participant A. The timings from single ports are not shown, as they are almost negligible with a maximum of 2 ms.

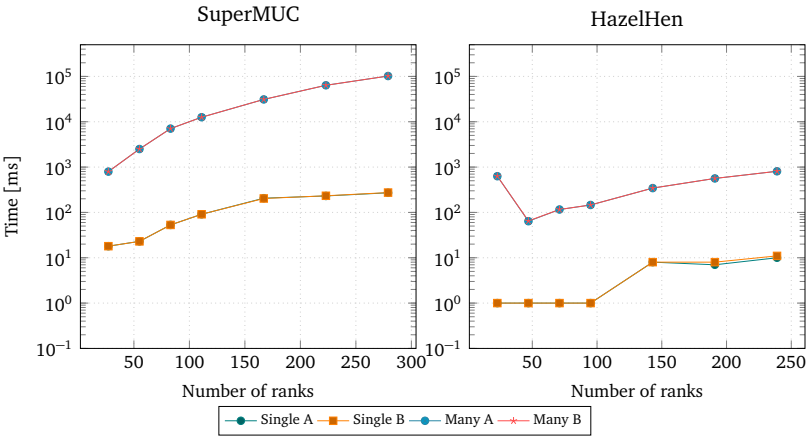


Figure 5.5: Establishing the connection between the participants using `MPI_Comm_accept` and `MPI_Comm_connect`.

to this, showing a degraded performance. The reason for this remains unknown. The performance of the HPC systems are almost on par, whereas in the previous phase, Fig. 5.5, the HazelHen system showed superior performance.

As a conclusion, it can be said, that the newly implemented design, using only one communicator shows a vastly improved performance in the two phases of connection establishment. For the actual data transfer, both variants are on par, with the noted exception. This exception can probably be attributed to a singular performance degradation, due to unfortunate setting of parameters by the operator exhibited by an unrealistic small amount of data to be transferred.

5.4.2 Communication Decoupling of Participants

While most multi-physics simulations are two-way coupled simulations, one-way coupling scenarios can also exist, e.g, a fluid simulation coupled to an acoustic far-field simulation. From an algorithmic perspective, the sending participant (here the fluid simulation) can run completely independent from the receiving one. However, this concurrency is limited by interleaved send and receive operations in the implementation and the amount of asynchronicity provided. Figure 5.8 (top) visualizes the issue. The situation shown involves a fluid solver using a splitting approach, performing incompressible fluid dynamics, followed by a near-field acoustic simulation with sub-cycling inside one timestep. Such a splitting approach is implemented in the solver FASTEST [72] and used in the fence test case, Fig. 5.7. The near-field accoustics solver performs a uni-directional coupling with a far-field acoustics solver and sends coupling data each sub-cycle step. Without buffering, each send operations can only be completed, when the far-field solver has reached the same sub-cycle and is ready to receive the results. Careful load-balancing avoids idle times in participant A by matching the time consumed by the near-field acoustics (FA) and far-field acoustics (AFF) solvers. Idle times in solver B, however, can not be avoided this way. A buffer, as shown in Fig. 5.8 decouples the two solvers and this way prevents idle times and allows for more flexible assignment of computational resources.

This issue is alleviated by the existence of internal buffers, either within the MPI implementation or from the operating systems TCP/IP stack. These internal buffers give the simulation a limited decoupling, even without explicit buffer management in the coupling library. This, however, is insufficient and unreliable. The size of internal buffers is opaque to the user and differs from system to system. In the case of TCP-buffers, their size can not be modified as a non-root user. While the buffer size of MPI implementations can be modified by the user, it is usually tuned by the vendor to hit a sweet spot of the machine

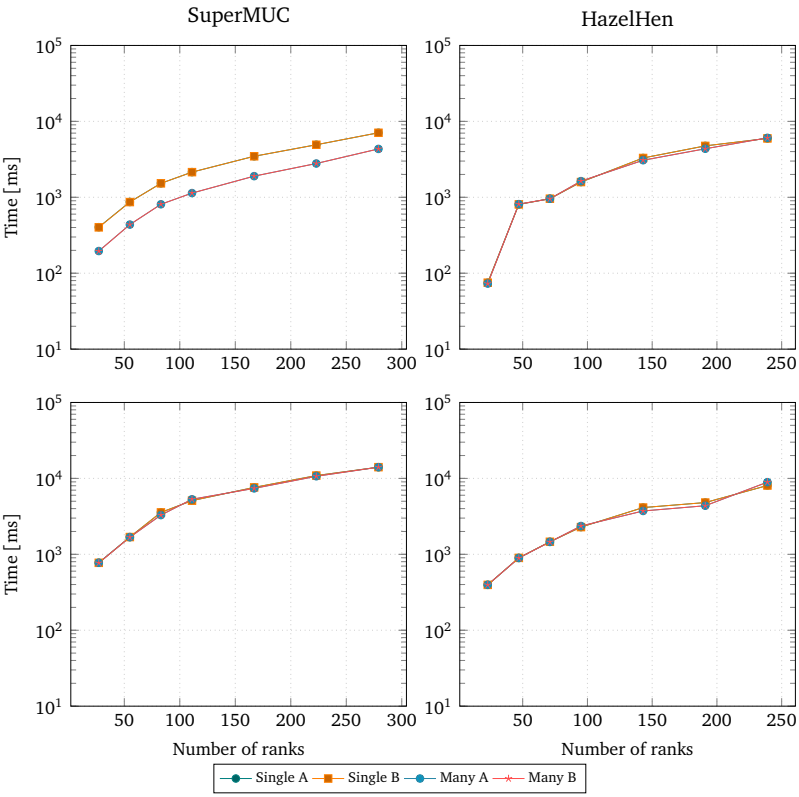


Figure 5.6: Times for 1000 rounds of a data transfer of a vector of 500 doubles (top row) respectively 2000 doubles (bottom row) from participant B to A. For the transfer the synchronous MPI routines (`MPI_Send` and `MPI_Recv`) have been used.

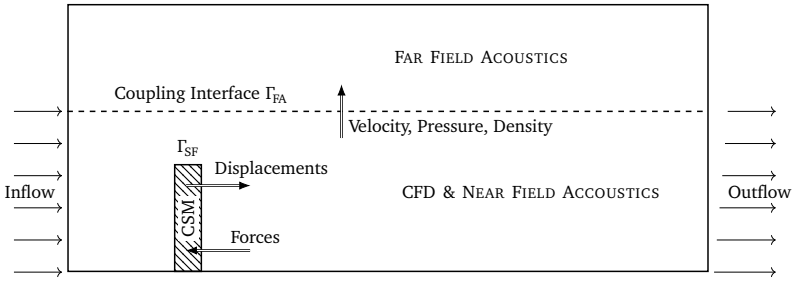


Figure 5.7: Fence test case with a splitted CFD/near-field acoustics setup coupled to structure and far-field acoustics solver. The CSM solver used is FEAP [100], the CFD and near-field acoustics solver is FASTEST and the far-field acoustics is computed by Ateles. Note, that the geometry is not to scale.

performance wise. In both cases, too large buffer sizes might result in a phenomenon which is known as bufferbloat [22] and might affect the overall performance of the simulation.

Both MPI or TCP sockets implementations provide fully asynchronous send operations. These operations take a pointer to a buffer and return a handler for the request. Using the handler, one can test for completion of the send operation and subsequently release the buffer. The obvious implementation strategy of such a buffer manager would be to create a separate thread, which periodically iterates through the stored requests and tests for completion. The sending thread adds the request handle and the pointer to the buffer to a shared data structure and proceeds.

The resulting multi-threading, however, requires special caution to avoid data races and other synchronization issues. This so called thread safety is not only needed in the main application but also in libraries that are used in a multi-threading context.

Points that need synchronization in a multi-threaded implementation to ensure correct ordering of buffers are handover of the handler and buffer, testing for completion and removal of the handler and buffer. The first and the latter are easy to synchronize, using mechanism such as a lock guard, the testing operations warrants closer inspection.

The implementation of socket communication inside preCICE is based on Boost Asio. It uses callback functions to signal the completion of an asynchronous request and thus requires no special synchronization.

MPI implementations are however much less transparent. According to [48, p. 12.4] MPI defines four levels of thread safety

THREAD_SINGLE Only one thread will execute

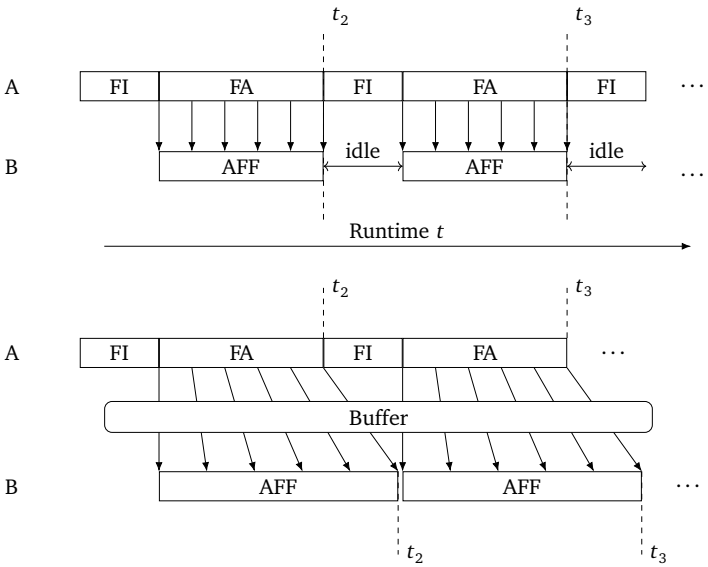


Figure 5.8: One way coupling scenario between participant A, performing incompressible fluid (FI), followed by near-field acoustic simulation (FA). The near-field acoustics solver performs subcyling and the result of each sub-step is transmitted to far-field acoustic solver (AFF). Top: Without buffering idle times for participant B are created. FA is bound to AFF though send operations, therefore, the runtimes of FA and AFF are matched through careful load-balancing. Bottom: A send buffer decouples fluid and acoustics solver for send operations, prevents idle times and allows for more flexible processor assignment.

System	Implementation	Default level	Highest level
ArchLinux	Open MPI 3.1.0	Single	Multiple
ArchLinux	Open MPI 4.0.1	Single	Multiple
ArchLinux	MPICH 3.3	Single	Multiple
Ubuntu 16.04	Open MPI 1.10.2	Single	Serialized
Ubuntu 16.04	MPICH 3.2	Single	Multiple
SuperMUC	Intel MPI 2017.4	Funneled	Multiple
SuperMUC	IBM MPI 1.4	Single	Multiple
HazelHen	Cray MPICH 7.70	Single	Serialized

Table 5.1: Highest provided and default level of thread safety of different MPI implementations / systems

THREAD_FUNNELED Multiple threads may be running, but only the main thread makes MPI calls

SERIALIZED MPI calls are allowed from multiple threads, but not concurrently, only one at a time

MULTIPLE Multiple threads are possible, with no restrictions

While one might think, that a call merely testing a request for completion is read-only on internal MPI data structures, each call to an MPI function triggers the progress engine and may as well initiate pending sending or receiving operations. The multi-threaded buffer manager requires the highest level of thread safety, which is `MPI_THREAD_MULTIPLE`.

This can become problematic due to two different reasons. Table 5.1 shows a sample of the default and highest level of thread safety different MPI implementations provide. Two of the tested implementations do not provide the highest level and would, thus, be unable to use the buffer manager or even to safely run preCICE as a whole.

Furthermore, the desired level of thread safety must be requested at initialization time by a call to `MPI_Init_thread`. In a typical simulation setup involving preCICE, the MPI initialization is done by the host application, i.e., the solver. preCICE has no influence on the kind of MPI initialization without additional cooperation from the adapter or the solver. Such a cooperation, however, is not desirable in terms of easy black-box integration. The default level provided, furthermore, is insufficient for all implementations.

Because of the problems outlined above, a non-threaded implementation was developed. The implementation consists of two parts:

(i) The higher-level part takes care of the memory management of the buffers. Data buffers which are sent asynchronously are saved in a list together with their respective

request. The requests saved in the list are regularly checked for the their status and the memory is freed if the data was sent. For reasons given above, no thread is used here and, instead, the checks are triggered at certain occasions, e.g., after a new asynchronous send operation was dispatched and before the communication is destructed.

(ii) The lower-level part is only relevant for TCP socket communication. Asynchronous sending operations can "overwrite" itself, i.e., a buffer which is not yet sent completely is then sent interleaved with other chunks of data that can originate from asynchronous or synchronous send operations initiated after the first. This obviously leads to mangled data at the recipient. For that reasons, asynchronous send operations on TCP sockets are not directly given to the TCP stack, but instead placed in a queue. Using a callback mechanism, a completed operation calls the next one in the queue. Likewise, a queue sweep is initiated whenever a new element has been added to it. MPI does not require such a mechanism, as it guarantees to preserve the order of messages.

The effectiveness of this change has been validated using the *fence* test case on the SuperMUC system, see Fig. 5.7 for description. The case was intentionally configured with a high load imbalance, giving 124 cores to FASTEST, performing the inner, fluid simulation and only 2 cores to Ateles for the outer, far-field acoustic simulation. Before the simulation was terminated after 30 minutes, the decoupled variant was able to perform 1963 coupling iterations compared to 55 iterations for the non-decoupled variant.

5.5 Conclusions

preCICE implements three different communication channels needed for distributed, partitioned multi-physics simulations. These three channels, which are MasterSlave, Master and M2N communication, are realized using either TCP or MPI in different variants.

During initialization, a large number of connection tokens must be passed from each rank at one participant to the other participant. To reduce the load on the file system caused by creating a large number of files containing connection information, a flexible and future proof scheme to distribute that information among directories and files has been presented.

MPI is the state of the art communication protocol in scientific computing, shining with its collective operations and hardware support from HPC vendors. However, preCICE uses significant parts of the dynamic process management features from the MPI standard. While testing on different machines, it has turned out that this part of the standard is problematic in practice. This ranges from crucial features not being supported to more subtle limitations such as severe performance degradation or limits in the number of ranks. For these reasons,

a mature TCP sockets implementation is, though inferior in theory, a must-have.

In preCICE, the MPI ports mirrored the TCP implementation, in so far that it used one handle (communicator or socket, respectively) for each connection. For MPI, this turned out to result in sub-optimal results performance wise. Using just one communicator that encompasses all ranks, even if a connection to some of the ranks is not required, results in superior performance.

The low-level characteristics of TCP raises distinct challenges. The lacking guarantees regarding message order are countered by implementing a message queuing system inside preCICE. This enables preCICE to fully asynchronously perform send operations, a use case that is motivated from a concrete issue that arose in the ExaFSA project, see Sec. 1.2.

6 Software Development

> Computer science research is different from these more traditional disciplines. Philosophically it differs from the physical sciences because it seeks not to discover, explain, or exploit the natural world, but instead to study the properties of machines of human creation. In this it is analogous to mathematics, and indeed the "science" part of computer science is, for the most part mathematical in spirit. But an inevitable aspect of computer science is the creation of computer programs: objects that, though intangible, are subject to commercial exchange.
>
> Dennis Ritchie

For a computer scientist, the software is not only a tool for research, but can be the objective of research itself.

In this chapter, I will present how the preCICE project tackles challenges, that arise with developing a partitioned, multi-physics simulation software, focused on large HPC installations.

Developing software is not only about technical aspects, but also a product of a team effort. The development process of scientific software differs substantially from that of commercial software. Commercial software is developed using requirements engineering and strong leadership. The explorative nature of scientific application development makes it impossible to apply a strict process of requirements elicitation. Furthermore, the involvement of different stakeholders such as students focusing on their thesis, principal investigators focusing on specific projects and domain specific scientists on a particular problem, often results in a highly distributed development process. For characterizations and models of scientific software development, see [5, 27, 94], or [76] for an literature overview.

In this section, I will concentrate on the technical aspects of certain parts of preCICE's development, mainly those of profiling and testing.

6.1 Profiling

Profiling is a form of program analysis that measures the application during runtime. Quantities that are measured can include but are not limited to memory consumption, hard disk or network input/output operations and frequency or duration of certain function calls. The retrieved data from the profiling run serves as a database for subsequent optimization of the application.

Profiling, together with rigorous systematic testing is a pre-requisite for ensuring high-quality and reliability of research software, in particular if developed in large and diverse teams. Leveraging the high computing power of modern HPC machines makes thorough profiling crucial for applications in the high-performance computing area.

Profiling is achieved by instrumenting either the program source code or its binary executable form using a tool called a profiler (or code profiler). Profilers may use a number of different techniques, such as event-based, statistical, instrumented, and simulation methods.

Sequential applications can usually be profiled using methods such as

Automatic instrumentation modifies the application's source code. This happens, using a preprocessor or by support from the compiler itself. Gprof [52] is a popular, though slightly outdated, example of compiler based instrumentation. It adds moderate overhead of five to thirty percent to the runtime of the program being profiled.

Manual instrumentation relies on the developer to choose the source code blocks and functions to be instrumented using statements from the respective programming language or using a preprocessor language. This is something every developer usually does in a more or less systematic manner, ranging from simple timing statements scattered in the code to elaborate frameworks.

Sampling measures the application's state at certain time intervals. It usually uses support of the CPU to count hardware events such as instructions executed or cache misses and does not rely on instrumentation. Whereas instrumentation can provide exact times, sampling is inherently a statistical approximation. `perf`[1] is a well known example of a sampling tool.

While manual instrumentation for a complete application is certainly too much effort, it allows the programmer to focus only on relevant parts and to add supplemental information, such as iteration counts or FLOPS spent.

[1] `https://perf.wiki.kernel.org/index.php/Main_Page`

Profiling distributed, parallel applications adds additional complexity to the task. While a single rank of a parallel application can be profiled the same way as a purely sequential application, the developer is usually also interested in the parallel behavior of the application, to identify dependencies between ranks, load imbalances or an overview over all ranks, something which can become challenging for high rank counts.

In order to support the developer as well as the user to perform manual instrumentation and analyse the output in a sustainable way, the *EventTimings*[2] package has been developed and integrated into preCICE. It is available as a stand-alone project, licensed under LGPL3 [51]. Requiring no external dependencies beyond the C++11 standard library and MPI makes integration into existing codes effortless.

It is partly inspired by the PETSc Logging subsystem, sharing the idea of an *Event* that can be started and stopped at certain locations in the code. On the other hand, by employing modern object oriented techniques based on the C++11 standard, it requires less boilerplate code.

The *EventTimings* framework is designed for software that is parallelized using the Message Passing Framework (MPI). Thus, at the end of a profiled run, data from all ranks are collected and timings are normalized, as detailed in Sec. 6.1.1.

The overhead of profiling and data acquisition is small enough, so that the profiling code could stay in place for production runs.

Listing 6.1 shows the basic initialization and finalization of the framework.

```
1 EventRegistry::instance().initialize(appName, runName, mpiComm);
2 [...]
3 EventRegistry::instance().finalize();
4 EventRegistry::instance().printAll();
```

Listing 6.1: EventTimings initialization with optional parameters and de-initialization

During initialization an optional `appName` and `runName` can be given. The first identifies the participant and influences the naming of the output files. The latter can be freely chosen. It is written to the result files and can be used to identify a particular test setup later. The MPI communicator used defaults to `MPI_COMM_WORLD`. It can be set to the one used by the application. Note, that all ranks that use events must be contained in the supplied communicator.

The core is the `EventRegistry` singleton, which holds an `RankData` instance, storing the information for all events of a particular rank. `EventData` instances are used to store information for each distinctively named event. Information stored there is already aggregated,

[2] https://github.com/precice/EventTimings

e.g., minimum, maximum and average execution times. The principal components of framework are shown in Fig. 6.1.

```
Event e("Name", barrier, autostart);
// timed code
e.pause();
// non-timed code
e.start();
// timed code
e.addData("IterationNumber", iterationNumber);
e.stop();
```

Listing 6.2: Usage example for an Event, demonstrating creation, pausing and re-starting of an event, as well as adding supplemental data.

Listing 6.2 demonstrates the basic use of events. Events posses three states, which are *started*, *paused* and *stopped*. Pausing and restarting an event merely stops and restarts the timer, whereas stopping and restarting an event is counted as a new invocation of the event. All state changes are logged, collected from all ranks, normalized and written to a log file named `appName-events.json` upon finalization. The log file is formatted using the JavaScript Object Notation (JSON), which has evolved as a standard for lightweight hierarchical data formats. The format can easily be parsed for post-processing. The Python programming language includes a JSON parser in its standard library.

Additionally, arbitrary integer data can be appended to an event, such as an iteration number. The data are stored in a key-value store associated with an event name. Like the timings, they are written to the JSON file at the end of a run.

Synchronisation points In order to achieve maximal parallel efficiency, synchronization points, such as barriers, should be limited to a minimum. On the other hand, to gain insight into the parallel behavior of an application, it can be helpful to have explicit synchronisation points. For that, preCICE offers a configuration option to set a *sync mode*, enabling explicit synchronisation. Synchronisation points at the beginning and end of most events are then activated and work at the scope of the participant. Additionally, an inter-participant synchronisation is performed before the beginning of the send and receive operations of coupling data between participants. While synchronisation inside a participant can easily be achieved using a call to `MPI_Barrier`, inter-participant synchronisation is implemented using a three-way handshake, i.e., send-receive-send operations on the sender side and receive-send-receive on the receiver side.

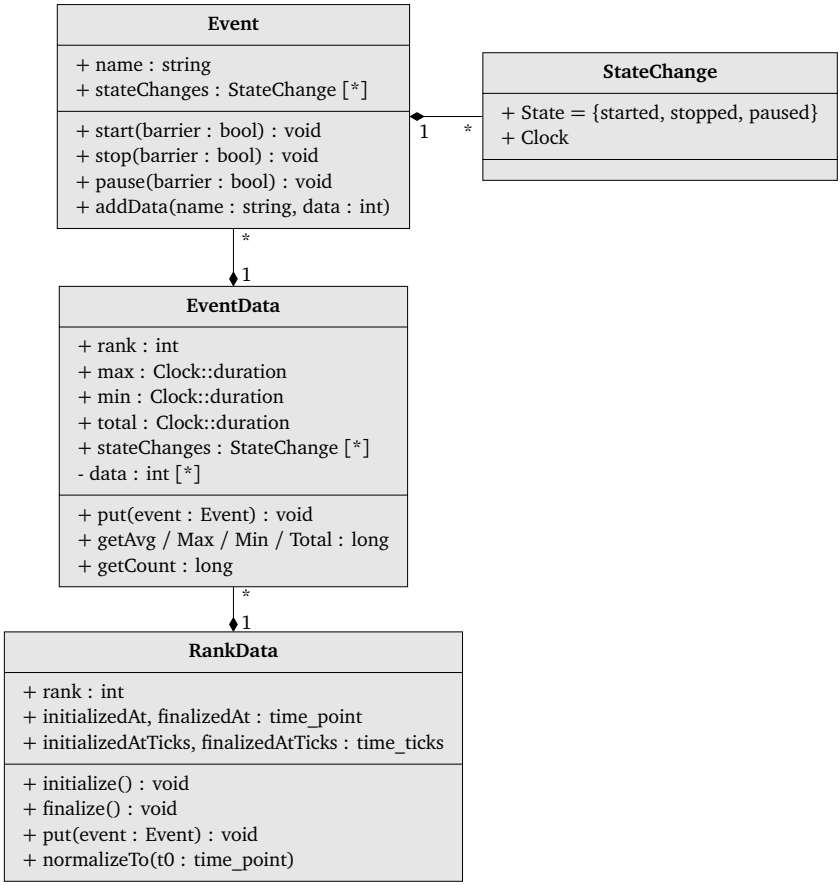


Figure 6.1: UML-like diagram of the principal components of the EventTimings library. An `Event` represents a single event and its state. After stopping the event it is aggregated in the `EventData` class, which stores data for one type of event at a particular rank. All event data at one rank are saved in the `RankData` class.

6.1.1 Measuring Time for Distributed Applications

Selecting a Suitable Clock Source

Performing exact timings on modern multi-tasking computers and correlating them between different machines is a challenging task. Some of the intricate complexity of establishing a common time base across a set of nodes in a distributed system is detailed, e.g., in [67].

Critical features of a clock are

Accuracy in terms of the official standard time as well as stability of the tick frequency. This can only be influenced by the underlying hardware and the operator thereof. In the following, I assume this criterion to be met.

Monotonicity guarantees that the time can never decrease as the physical time moves forward. A decrease in time can happen if the system clock is adjusted at intervals to maintain accuracy. Monotonicity is required for a reliable measurement of time durations.

Resolution gives the smallest time interval or tick that can be measured by the clock.

Absoluteness determines if points from two completely separate measurements can be correlated, i.e., they are related to a common universal zero-point. Separate here means, that they are done on different machines, from different processes or on a system that was rebooted between measurements.

Accuracy to the official standard time and monotonicity are contradicting feature requirements as, to keep accuracy, the clock needs to be re-adjusted periodically. Consequently, no available clock source can deliver all of these features.

This contradiction is also reflected in the low level clock sources, as offered by the POSIX functions for Linux systems. Clock sources available are either `CLOCK_REALTIME` or `CLOCK_MONOTONIC`. The latter measures the time since some unspecified starting point.

The system clock (also called wall-clock or real time clock) can be changed in both forward and backward direction anytime, e.g. by the NTP daemon. On the other hand, the monotonic clock acts like a stopwatch, which is suitable for measuring durations, but can not be correlated to a specific time point.

The C++ programming language introduced the `<chrono>` library in its 2011 revision [62], providing an object-oriented and type safe method to work with time points and durations. The design of the library is such that accuracy loosing conversions require explicit casting. Time points and durations are derived from a specific clock source to retain

its accuracy. The library offers two clock sources, a *system clock* and a *steady clock*. While the first meets the feature of absoluteness, the latter does not, but provides a monotonic clock, suited for duration measurements.

A much simpler approach is offered by the MPI's `MPI_Wtime` [48, section 8.6] function. It returns a double-precision number of seconds since some arbitrary time in the past, also called the *epoch*. This time point is guaranteed to not change during the life of a process. Therefore, this clock source provides monotonicity and sufficient resolution, but no absoluteness.

As both preCICE and the EventTimings framework are implemented in terms of C++11, the chrono library was chosen.

Normalizing measured times

The duration of an event is logged using the start and end time point of it, given by the steady clock. For a useful interpretation of the measurements over multiple ranks and participants, it is required to absolutely correlate the start and end time points of events. Since the steady clock's epoch can be different for each machine, they cannot be related to the system clock without further reference.

For every rank, at the beginning of the run, the system clock and steady clock times, t_0 and $\tilde{t}_0$ are recorded. These times can later be used to normalize the timestamps to a common t_0. This t_0 is chosen as the first, i.e., minimum starting time over all ranks, as shown in Alg. 6.

Data: Ranks R, Timestamps S
Result: Normalized time stamps $s \in S$
1 $t_0 \coloneqq \min_{r \in R} t_{0,r}$
2 **for** $r \in R$ **do**
3 $\quad \delta \coloneqq t_{0,r} - t_0$
4 $\quad$ **for** $s \in S_r$ **do**
5 $\quad\quad s \leftarrow s - \tilde{t}_{0,r} + \delta$
6 $\quad$ **end**
7 **end**

Algorithm 6: Time normalization among ranks of one participant of a coupled simulation. Time stamps relative to the steady clock are denoted as $\tilde{t}$. Line 1 can be implemented as an MPI reduce collective operation. The following lines are rank-local and can be executed in parallel.

Algorithm 6 is employed right after a run and normalizes the times for all ranks of a

single participant in a coupled simulation. In a partitioned coupling scenario, however, there is the need to also correlate times of events from different participants, stemming from two or more different log files. As a post-processing step, a second normalization is employed. Very similar to the rank-scoped normalization, Alg. 7 shows the normalization among the participants.

Data: Participants P, Timestamps S
Result: Normalized time stamps $s \in S$
$t_0 := \min_{p \in P} t_{0,p}$
for $p \in P$ **do**
  $\delta := t_0 - t_{0,p}$
  for $s \in S_p$ **do**
    $s \leftarrow s + \delta$
  end
end

Algorithm 7: Time normalization among multiple participants. Ranks of one participant are already normalized by means of the previous algorithm.

As this algorithm works on the log files, the accuracy is limited to the resolution used in the log files, which is currently milliseconds. Algorithm 6, on the other hand, works on chrono library objects and therefore retains the maximal accuracy available.

6.1.2 Visualization

The insight that can be gained from data acquired during a run is not only related to correctness and the amount of data. A coupled parallel simulation can generate tens of thousands of events, resulting in huge log files, which are incomprehensible without proper visualization. We use the *trace-viewer*[3], a JavaScript application that powers the front end of Chrome's `about:tracing` interface and the Android systrace. The application also uses the concept of events that are recorded when started or stopped and its data model maps nicely with the data generated by the EventTimings framework. The EventTimings output files for a single participant or from a run with multiple participants can be converted into one trace file in the trace-viewers format[4]. For that, a utility named `events2trace.py` is provided. This post-processing step also performs the time normalization as shown in Alg. 7.

[3] `https://github.com/catapult-project/catapult/tree/master/tracing`
[4] `https://docs.google.com/document/d/1CvAClvFfyA5R-PhYUmn5OOQtYMH4h6I0nSsKchNAySU`

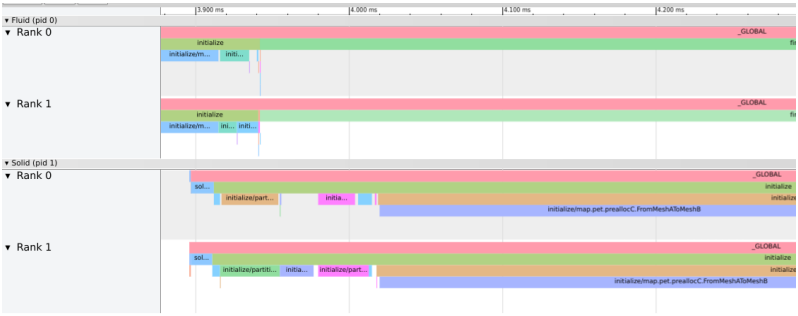


Figure 6.2: Screenshot of EventTimings visualization using the trace-viewer. It shows the startup of a coupled simulation consisting of two participants, “Fluid” and “Solid”.

Figure 6.2 shows the events at the beginning of a coupled simulation, with the times being normalized according to the aforementioned algorithms. Using the trace-viewer allows to pan and zoom freely and thus to analyze events ranging from durations of milliseconds to hours. It also allows to select events inside a certain time slice and query them for statistics, such as average times, or the time that this event is active exclusively.

To facilitate using a wider range of visualization tools, the Open Trace Format Version 2 (OTF2)[5] could be added as an output format in the future. The OTF2 format is used by various post-processing tools, such as Scalasca Cube[6] or the Intel Trace Analyzer[7]. OTF2 uses the a similar data model, also driven by the concept of events. However, the format is much more powerful, including record types, such as for I/O and MPI send or receive operations.

6.2 Testing

Software testing is well established and recognized as a good practice of software development. While this is also true for scientific software, this type of software holds special challenges to software testing [57]. The exploratory nature of scientific programming makes using a test driven approach hard, since test results are defined as a formal set of requirements. Whereas in a scientific process, they are not known a priori, but established

[5] https://www.vi-hps.org/projects/score-p/
[6] http://scalasca.org/software/cube-4.x/
[7] https://software.intel.com/en-us/trace-analyzer

and refined iteratively. These kinds of problems are called oracle problems. Kanewala and Bieman [65] identify the oracle problems as one of the two main challenges in testing scientific software. Cultural differences between domain scientists and software engineers is identified as the second main challenge. preCICE can be perceived as an effort to overcome the latter challenge. The partitioned coupling approach can help to maintain a separation of concerns between the solver or simulation experts and the developers of the coupling library. Furthermore, the partitioned approach also enables to consider the participating solvers as independent instances and, thus, leads to improved testability.

Studies about defect density claim a rate of about 20 to 50 errors per 1000 lines of code [40, 74] in general software projects. Scientific software projects are frequently not developed by computer scientists or trained software developers but by scientists working in a particular domain, such as physicists or chemists. This gives reason, that the error rate in scientific software is even higher than in commercial projects.

Automatic software tests are one step to reduce the error rate in software. They can be embedded in a continuous integration (CI) process [37]. A CI server is the core of a continuous integration system. It builds and tests the software in a reproducible manner after each source code change. preCICE uses Travis CI[8], a cloud-based CI service that can be used at no charge from open-source projects.

In this section, I will address the special challenges a parallel coupling software faces when developing unit and system tests. For a general introduction to testing, I refer to *Continuous Integration: Improving Software Quality and Reducing Risk* [37].

6.2.1 Unit Tests

Unit tests are tests of the smallest functional units of a software, i.e., single functions or classes. Unit tests consist of *test cases* that are arranged in *test suites*. Test suites can be nested in a hierarchical manner. Each test case tests one functionality using one or more assertions. Tests that require certain pre-conditions can declare a *fixture* to setup and tear down prerequisites of the tests. To ease the development of unit tests, the *Boost Test*[9] framework is used for preCICE.

Much of the functionality contained in preCICE can not be tested on a single processor but only in parallel. We use so-called *decorators* to express requirements regarding the parallel execution of tests in a declarative manner.

[8] `https://travis-ci.org`
[9] `https://www.boost.org/doc/libs/1_69_0/libs/test/doc/html/index.html`

Single Rank declares that the MPI communicator used by this test should be restricted to only contain a single processor. The test itself, however, is still executed on all processors, albeit each in their own disjoint communication space. A use case is a MPI aware function that should be tested on a single processor.

On Ranks allows to restrict the execution of the test to certain processors only, e.g., for testing a function on a specific number of processors.

On Master likewise restricts the execution to processor zero. This can be used for non-parallel code that should not be run independently on multiple ranks, e.g, because it performs input/output operations.

Min Ranks declares the minimum size of the communicator required by the test to run. If the test binary is executed on an insufficient number of processors, the test is skipped.

While the declarations above restrict the execution of tests to certain conditions or modify the communication environment, fixtures can be used to establish pre- or post-conditions for the tests. They can be used to setup and tear-down connections before and after a test, respectively. To understand the different kinds of communication channels preCICE uses, please refer to Chapter 5.

Sync Processors forces the processors to sync before and after a test, e.g., using an `MPI_Barrier`. This is especially useful to keep the output of the test run on different ranks closely together.

MPI Comm Restrict accepts a vector of ranks and creates an MPI communicator that only consists of the given ranks. After the test, the global communicator is reset to its former state.

Master Com Fixture establishes a communication between processor 0 as *master* and all other processors as *slaves*, used for, e.g., testing the re-partitioning of a received mesh inside one participant.

M2N Fixture creates an M2N connection between two participants. The fixture declares an instance of `com::M2N` that can be used by the test to access the communication.

Split Participants Fixture splits the communicator into two groups, without establishing a communication. The first group consists of the former rank 0, the second group contains all other ranks.

To accurately describe the requirements regarding parallel execution, usually multiple declarations are combined, e.g., to establish a master-slave communication the test obviously requires a minimum of two processors. Listing 6.3 shows an example of multiple decorators to setup suitable parallel testing conditions.

```
BOOST_FIXTURE_TEST_CASE(TestName, testing::M2NFixture,
   * testing::MinRanks(2)
   * boost::unit_test::fixture<
         testing::MPICommRestrictFixture>(std::vector<int>({0, 1})))
{
// test code that needs an established M2N connection
BOOST_TEST(m2n.isConnected() == true);
}
```

Listing 6.3: Unit test example with decorators that declares a minimum number of ranks, a fixture to create and tear-down a M2N connection and restricts the MPI communicator to only contain ranks 1 and 2

In addition to unit tests, which test the smallest functional units of a software, there are also *integration tests*. Integration tests work by combining the basic functionality into higher level tests. As an example, a unit test uses three test cases to test if a mesh correctly supports adding and deleting vertices as well as merging with another mesh. The integration test, on the other hand, creates two participants with meshes, connects them and performs a mapping between the two participants. In preCICE, integration tests are implemented similar to unit tests and can use the same set of decorators and fixtures.

6.2.2 System Tests

While the unit and integration tests from the previous section look at the tests from a developer perspective, system tests perform the testing from a user perspective, i.e., the system is tested the same way a user would use it.

As preCICE is a library to couple single-physics solvers, these kind of tests require a minimum of two applications (here, preCICE does not count as a application, since it is linked to the solver and therefore becomes part of the solver's process space). To provide a constant and reliable environment for compilation and execution, we use the Docker[10] container platform. Docker provides operating system level virtualization for software packages, called containers, isolated from each other. Containers are instantiated from images (the image – container relationship is much like the object – class relationship in object-oriented programming). Images are created by shell-script like instructions from a `Dockerfile` and can be derived from already existing images.

[10] `https://www.docker.com`

The first version of the system test framework[11] was developed by Hoshaber as part of his Bachelor thesis [59]. The principal steps of a test run, following a *build-run-compare* scheme are

1. Creation of the preCICE base image from the `Dockerfile` by cloning the repository and build preCICE from source.
2. Based on this image, the actual solver image is created. It contains both solvers and the test data, i.e., the geometry and description of the case that is going to be simulated.
3. Running the simulation. At the end of the simulation, the result data are copied into a designated directory inside the container.
4. Copying the results from the container and comparing it with the given reference data.
5. A report of the test outcome is uploaded to a separate repository. In case of a failing test, the generated results are also included.

Currently, test cases for three different open-source solvers are implemented. OpenFOAM [109] is used for fluid and conjugate heat-transfer simulations. Calculix [33] is a free three-dimensional structural finite element program. It is coupled with the SU2 [39] fluid-simulation code.

Using these solvers, three system test cases exist:

OpenFOAM - Calculix is a fluid–heat-transfer simulation of a shell-and-tube heat exchanger. It features a complicated geometry and laminar flow. The case can also be found as part of the preCICE tutorials[12]. See Sec. 4.10 for a more information on that case.

OpenFOAM - OpenFOAM simulates a conjugate heat-transfer between a plate and a fluid. OpenFOAM is used for both participants, simulating the fluid flow and the temperature distribution in the plate.

SU2 - Calculix models a moving fluid in a channel that actuates a flexible flap.

[11] `https://github.com/precice/systemtests`
[12] `https://github.com/precice/tutorials`

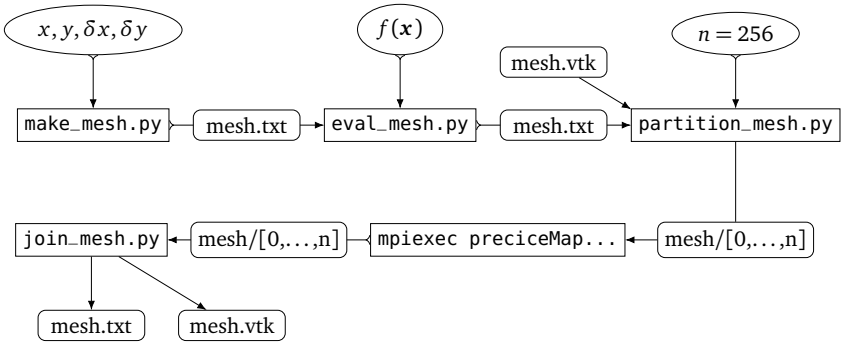


Figure 6.3: Schematics and workflow of the ASTE test suite, showing creation of a mesh, evaluating a function on it, execution and reconstruction of the result mesh. Executables are shown in rectangles, data in rectangles with rounded corners, input in ellipses.

The system tests are integrated into the continuous integration system. Because the computational load by the compilation of the solvers and the subsequent simulation runs are considerably high, the system tests are not run on every commit. Instead, they are are executed on a daily basis on the recent development branch of preCICE.

6.2.3 ASTE – A Flexible Coupling Test Framework

ASTE[13] (dubbed *Artificial Solver Testing Environment*) is a suite of tools that are developed to ease the testing of preCICE. As a library, preCICE itself is intended to be used by other applications. ASTE uses the official API of preCICE and behaves like a solver to preCICE. To be flexible and adaptable, ASTE is developed as a suite of small applications, most of them written in the Python programming language. Figure 6.3 shows a workflow how a mesh can be created with values from a function evaluated on it and partitioned to the target number of processes. After execution of `preciceMap` which is the solver to preCICE, the output mesh can be reconstructed and analyzed.

`make_mesh.py` can be used to generate a rectangular uniform mesh in two dimensions with given extension in x- and y-direction and a given number of elements in each direction. The mesh is technically three-dimensional, but its extension in z-direction is 0. The mesh is saved as a textual representation in the format `x y z v`, where v is a scalar value for the

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

particular vertex, initially set to zero. The value v can be set by executing `eval_mesh.py`, which is supplied a function $f(x, y, z)$, evaluated on each vertex. v represents the value that is send/mapped to the other participant via preCICE.

`partition_mesh.py` decomposes the mesh into a given number of partitions. It uses the VTK toolkit [93], which enables it to read in a range of mesh formats, including, but not limited to, variants of VTK, STL (stereo-lithography) meshes, PLY Polygon File Format [104] and, of course, the textual format generated by `make_mesh.py`. To achieve a suitable decomposition, different algorithms, based on the existence of mesh connectivity information and the geometry of the mesh can be used. Some mesh formats contain cell connectivity information. This enables the use of graph-based partitioning algorithms, such as METIS [66]. If no such information is available, k-means clustering can generate a suitable decomposition for meshes with a challenging geometry. The outcome of the k-means algorithm heavily depends on the initial placement of k centroids that form the centers of the clusters. In some cases, the method can return empty clusters, i.e., processors to which are no vertices are assigned. A warning is issued in such a case. A third method of decomposition is meant to be used on uniform, quasi two-dimensional meshes, such as the ones generated by ASTE. It provides a predictable decomposition into uniform, rectangular partitions. Figure 6.4 compares the outcome of the uniform and k-means partitioning.

The partitioned mesh is read by `preciceMap`, which implements a minimal client to preCICE. It is given the mesh directory and the participant, which can be either A or B, and the XML configuration file of preCICE. As the application is focused on testing the interpolation implemented in preCICE, it only does a mapping from A to B and writes out the received values. The output mesh directory can be transformed into either a VTK file or the textual mesh representation using `join_mesh.py`.

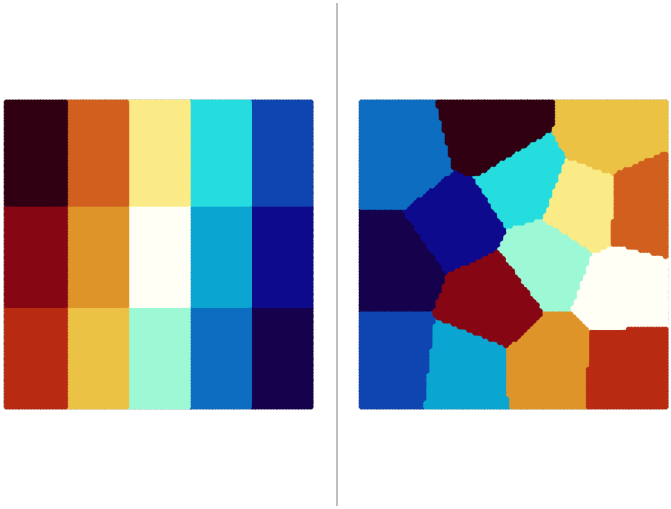


Figure 6.4: Different decompositions of a 100 × 100 mesh to 15 partitions. Left: Uniform partitioning, right k-means clustering

7 Conclusions

> There is no real ending. Its just the place where you stop the story.
>
> Frank Herbert

This thesis touches two, seemingly disconnected, fields: Interpolation and communication in the context of partitioned, black-box multi-physics simulations.

The interpolation with radial-basis functions proved to be a simple, yet powerful interpolation method. Compared with other interpolation methods, it stands out by requiring only minimal information from the input and output meshes. That makes it an ideal choice for interpolation in black-box multi-physics scenarios. Still, interpolation in the aforementioned context faces special challenges for the RBF method. Neither input nor output points can be chosen by the algorithm but are given and fixed by the coupling participants. Many extensions of the RBF interpolation assume that either one or both point sets can be chosen by the algorithm. This makes these kinds of extensions to the algorithm, e.g., multi-scale interpolation and adaptive methods that add or remove points based on a local error estimate not an option. The optimal support radius, as one of the most important parameters of the RBF algorithm is tightly linked to the mesh density. On non-uniform meshes, the choice of the basis function's shape parameter is always a trade-off between the dense and coarse parts of the mesh. For measuring the error of conservative interpolation a new error metric is presented and successfully used in the evaluation. Despite the shortcomings in accuracy due to error saturation, localized Gaussian basis functions with a support radius of $m = 6 \ldots 8$ provide a good compromise between stability and accuracy if combined with additional measures such as re-scaling as implemented in this thesis. The insights gained from the theoretical and experimental evaluations led to a highly-optimized algorithm for RBF interpolation. It is tailored for the very specific requirements that come with preCICE's re-partitioning strategy and the black-box approach as such. I integrated a tree index that allows for fast spatial queries and, thus to perform an efficient construction of the interpolation matrices. The developed parallel algorithm proved to scale well and its clean implementation provide a viable base for future extensions.

The second field, communication, involves setting up communication networks for

transmitting data between processors of one participant as well as between different participants. In an ideal world, MPI would be the only protocol required for communication, as it is tailored for HPC applications. Solver applications potentially bring their own implementation of MPI or alike with them and are tied to it, potentially not allowing a library to integrate with it. Furthermore, the implementation status of MPI regarding missing parts in the implementation of the required dynamic process management brings additional uncertainty. The topic of communication therefore contrasts interpolation by being less of a mathematical and algorithmic topic but tightly connected to the architecture of existing software and hardware systems. The newly implemented version of the MPI Ports communication together with the new scheme to exchange the initial connection information shows great improvements for the initialization phase and thus paves the way to even higher number of processors. The additional buffering in the TCP/IP implementation contributes to making the high-level communication of preCICE independent of the lower-level parts and to make TCP/IP "feel" more like the more sophisticated MPI. The decoupling of participants omits idle times that are unnecessary from an algorithmic point of view.

Upon closer inspection, however, interpolation and communication show significant overlap in the way they are used within preCICE. The layout of the parallel data structures in the implemented algorithm is determined by the layout of the respective participants. Communication, on the other hand, is steered by the data requirements of the interpolation. The uncertainty that comes with not being in control of the parallel layout requires careful implementation of the interpolation and communication by handling special cases, such as empty processors or disconnected partitions on one processor.

Partitioned simulations also provide new challenges with respect to profiling, debugging and testing. To profile preCICE on thousands of processors the *EventTimings* framework has been developed. It can effortless be integrated into other codes and the generated structured output files can easily be post-processed. For that reason, it is designed as a project independent from preCICE. In addition, the unit and integration test framework has been extended in this work to allow tests to express the requirements regarding parallel execution in a declarative manner. Furthermore, the system tests framework ensures easy testability and reproducible results of multi-solver configurations.

The preCICE project itself is on the best way to become a self-sustainable and healthy open-source project. The investment into clean code together with automated and thorough testing pays off. A growing number of adapters to different solvers are maintained within the project or as a part of the solvers itself. The mailing list and support channels reflect a growing number of active users who employ preCICE in their multi-physics simulations.

Bibliography

[1] Bob Alverson et al. *Cray XC Series Network*. Tech. rep. Cray Inc., 2012, pp. 1–28.

[2] E. Anderson et al. *LAPACK Users' Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[3] Satish Balay et al. *PETSc Users Manual*. Argonne National Laboratory, 2017.

[4] Satish Balay et al. *PETSc Web page*. `http://www.mcs.anl.gov/petsc`. 2016.

[5] Roscoe A Bartlett. "Integration strategies for Computational Science & Engineering software". In: *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. Vancouver, BC, Canada: IEEE, May 2009, pp. 35–42.

[6] Armin Beckert and Holger Wendland. "Multivariate interpolation for fluid-structure-interaction problems using radial basis functions". In: *Aerospace Science and Technology* 5.2 (Feb. 2001), pp. 125–134.

[7] Norbert Beckmann et al. "The R*-tree: an efficient and robust access method for points and rectangles". In: *SIGMOD Rec.* 19.2 (1990), pp. 322–331.

[8] C. Bernardi, Y. Maday, and A. T. Patera. "Domain Decomposition by the Mortar Element Method". In: *Asymptotic and Numerical Methods for Partial Differential Equations with Critical Parameters*. Ed. by Hans G. Kaper, Marc Garbey, and Gail W. Pieper. Dordrecht: Springer Netherlands, 1993, pp. 269–286.

[9] David Blom et al. "A Review on Fast Quasi-Newton and Accelerated Fixed-Point Iterations for Partitioned FluidStructure Interaction Simulation". In: *Advances in Computational Fluid-Structure Interaction and Flow Simulation*. Ed. by Yuri Bazilevs and Kenji Takizawa. Springer International Publishing, 2016, pp. 257–269.

[10] Aukje de Boer, Alexander 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 (Sept. 2008), pp. 4284–4297.

[11] Aukje de Boer, Alexander van Zuijlen, and Hester Bijl. “Review of coupling methods for non-matching meshes”. In: *Computer Methods in Applied Mechanics and Engineering* 196.8 (Jan. 2007), pp. 1515–1525.

[12] John P. Boyd. “Error saturation in Gaussian radial basis functions on a finite interval”. In: *Journal of Computational and Applied Mathematics* 234.5 (2010), pp. 1435–1441.

[13] John P. Boyd. “Six strategies for defeating the Runge Phenomenon in Gaussian radial basis functions on a finite interval”. In: *Computers & Mathematics with Applications* 60.12 (Dec. 2010), pp. 3108–3122.

[14] John P. Boyd and Lauren R. Bridge. “Sensitivity of RBF interpolation on an otherwise uniform grid with a point omitted or slightly shifted”. In: *Applied Numerical Mathematics* 60.7 (2010), pp. 659–672.

[15] John P. Boyd and Kenneth W. Gildersleeve. “Numerical experiments on the condition number of the interpolation matrices for radial basis functions”. In: *Applied Numerical Mathematics* 61.4 (2011), pp. 443–459.

[16] John P. Boyd and Lei Wang. “Asymptotic coefficients for Gaussian radial basis function interpolants”. In: *Applied Mathematics and Computation* 216.8 (2010), pp. 2394–2407.

[17] Markus Brenk. “Algorithmische Aspekte der Fluid-Struktur-Wechselwirkung auf kartesischen Gittern”. PhD Thesis. University of Stuttgart, 2007.

[18] I. N. Bronstein et al. *Taschenbuch der Mathematik*. 6. Frankfurt am Main: Verlag Harri Deutsch, 2005.

[19] Martin Buhmann. “Radial basis functions”. In: *Acta Numerica* 9.January 2000 (2000), pp. 1–38.

[20] Martin D. Buhmann. *Radial Basis Functions - Theory and Implementations*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2003.

[21] Hans-Joachim Bungartz et al. “preCICE - A Fully Parallel Library for Multi-Physics Surface Coupling”. In: *Computers and Fluids* 141 (2016), pp. 250–258.

[22] Vint Cerf et al. “BufferBloat: What’s wrong with the internet?” In: *Communications of the ACM* 55.2 (Feb. 2012), p. 40.

[23] Tony F. Chan and Henk A. Van der Vorst. “Approximate and Incomplete Factorizations”. In: *Parallel Numerical Algorithms*. Ed. by David E. Keyes, Ahmed Sameh, and V. Venkatakrishnan. Dordrecht: Springer Netherlands, 1997, pp. 167–202.

[24] Ward Cheney and David Kincaid. *Numerical Mathematics and Computing*. Cengage Learning, 2012.

[25] King Lum Cheung and Ada Wai-chee Fu. “Enhanced Nearest Neighbour Search on the R-tree”. In: *SIGMOD Rec.* 27.3 (1998), pp. 16–21.

[26] Davide Cinquegrana and Pier Luigi Vitagliano. “Fluid Structure Interaction Problems with CIRA structured CFD solver”. In: *6th European Conference on Computational Mechanics (ECCM 6), 7th European Conference on Computational Fluid Dynamics (ECFD 7)*. Ed. by Roger Owen et al. June. Glasgow: International Center for Numerical Methods in Engineerin (CIMNE), 2018.

[27] Carlton A Crabtree et al. “An empirical characterization of scientific software development projects according to the Boehm and Turner model: A progress report”. In: *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. IEEE, May 2009, pp. 22–27.

[28] Stefano De Marchi, Andrea Idda, and Gabriele Santin. “A rescaled method for RBF approximation”. In: *Springer Proceedings in Mathematics and Statistics* 201 (2017), pp. 39–59. arXiv: `1611.10034`.

[29] Stefano De Marchi and Gabriele Santin. “A new stable basis for radial basis function interpolation”. In: *Journal of Computational and Applied Mathematics* 253 (Dec. 2013), pp. 1–13.

[30] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 8200. Internet Engineering Task Force, July 2017.

[31] Simone Deparis, Davide Forti, and Alfio Quarteroni. “A Rescaled Localized Radial Basis Function Interpolation on Non-Cartesian and Nonconforming Grids”. In: *SIAM J. Sci. Comput.* 36.6 (2014), A2745–A2762.

[32] David Detlefs, Al Dosser, and Benjamin Zorn. “Memory allocation costs in large C and C++ programs”. In: *Software: Practice and Experience* 24.6 (1994), pp. 527–542.

[33] Guido Dhondt. *The Finite Element Method for Three-Dimensional Thermomechanical Applications*. Wiley, 2004.

[34] Tobin A. Driscoll and Bengt Fornberg. “Interpolation in the limit of increasingly flat radial basis functions”. In: *Computers and Mathematics with Applications* 43 (2002), pp. 413–422.

[35] Tobin A. Driscoll and Alfa R.H. Heryudono. “Adaptive residual subsampling methods for radial basis function interpolation and collocation problems”. In: *Computers and Mathematics with Applications* 53.6 (2007), pp. 927–939.

[36] Maksymilian Dryja and Olof Widlund. *An additive variant of the Schwarz alternating method for the case of many subregions*. Tech. rep. New York: New York University, Courant Institute of Mathematical Sciences, 1987.

[37] Paul Duvall, Stephen M. Matyas, and Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Signature Series. Upper Saddle River, NJ: Addison-Wesley, 2007.

[38] D. Eastlake and P. Jones. *US Secure Hash Algorithm 1 (SHA1)*. RFC 3174. Internet Engineering Task Force, Network Working Group, Sept. 2001.

[39] Thomas D. Economon et al. “SU2: An Open-Source Suite for Multiphysics Simulation and Design”. In: *AIAA Journal* 54.3 (2015), pp. 828–846.

[40] Nancy Eickelmann and Animesh Anant. “Statistical process control: what you don’t measure can hurt you!” In: *IEEE Software* 20.2 (Mar. 2003), pp. 49–51.

[41] Gregory E. Fasshauer and Jack G. Zhang. “On choosing optimal shape parameters for RBF approximation”. In: *Numerical Algorithms* 45.1-4 (Aug. 2007), pp. 345–368.

[42] Gregory E. Fasshauer and Jack G. Zhang. “Preconditioning of Radial Basis Function Interpolation Systems via Accelerated Iterated Approximate Moving Least Squares Approximation”. In: *Progress on Meshless Methods*. Ed. by A. J. M. Ferreira et al. Dordrecht: Springer Netherlands, 2009, pp. 57–75.

[43] Gregory F. Fasshauer. *Meshfree Approximation Methods with MATLAB*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2007.

[44] R. Fletcher. “Conjugate gradient methods for indefinite systems”. In: *Numerical Analysis*. Ed. by G. Alistair Watson. Springer Berlin Heidelberg, 1976, pp. 73–89.

[45] Bengt Fornberg, Elisabeth Larsson, and Natasha Flyer. “Stable Computations with Gaussian Radial Basis Functions”. In: *SIAM Journal on Scientific Computing* 33.2 (Jan. 2011), pp. 869–892.

[46] Bengt Fornberg and Cécile Piret. “A Stable Algorithm for Flat Radial Basis Functions on a Sphere”. In: *SIAM Journal on Scientific Computing* 30.1 (2007), pp. 60–80. arXiv: 1302.5877.

[47] Bengt Fornberg and Julia Zuev. “The Runge phenomenon and spatially variable shape parameters in RBF interpolation”. In: *Computers & Mathematics with Applications* 54.3 (Aug. 2007), pp. 379–398.

[48] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 3.1*. Stuttgart: High-Performance Computing Center Stuttgart, Aug. 2015, p. 836.

[49] Edgar Gabriel et al. “Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation”. In: *Proceedings, 11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary, Sept. 2004, pp. 97–104.

[50] Bernhard Gatzhammer. “Efficient and Flexible Partitioned Simulation of Fluid-Structure Interactions”. PhD Thesis. Technische Universität München, 2014, p. 261.

[51] *GNU Lesser General Public License*. Version 3. Free Software Foundation, June 29, 2007.

[52] Susan L. Graham, Peter B. Kessler, and Marshall K. Mckusick. “Gprof: A Call Graph Execution Profiler”. In: *SIGPLAN Not.* 17 (1982), pp. 120–126.

[53] Diane Greene. “An implementation and performance analysis of spatial data access methods”. In: *The Fifth International Conference on Data Engineering* 4871 (1989), pp. 606–614.

[54] William Gropp. “MPICH2: A New Start for MPI Implementations”. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Ed. by Dieter Kranzlmüller et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 7–7.

[55] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[56] Antonin Guttman. “R-Trees: A Dynamic Index Structure for Spatial Searching”. In: *SIGMOD Rec.* 14.2 (1984), pp. 47–57.

[57] Jo Erskine Hannay et al. “How do scientists develop and use scientific software?” In: *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. IEEE, May 2009, pp. 1–8.

[58] Jan S. Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin Methods*. Texts in Applied Mathematics. Springer, 2008.

[59] Yakup Hoshaber. “Entwurf und Implementierung von Systemtests für eine verteilte Multi-Physik Simulationssoftware”. Bachelor Thesis. Universität Stuttgart, 2018.

[60] University of Southern California Information Sciences Institute. *Internet Protocol*. RFC 791. Internet Engineering Task Force, Sept. 1981.

[61] University of Southern California Information Sciences Institute. *Transmission Control Protocol*. RFC 793. Internet Engineering Task Force, Sept. 1981.

[62] International Organization for Standardization (ISO). *ISO/IEC 14882:2011 Standard for Programming Language C++ (Working Draft)*. 2011, p. 1334.

[63] Telecommunication Standardization Sector of ITU. *Open Systems Interconnection - Model and Notation*. ITU-T Recommendation. Internation Telecommunication Union, 1994.

[64] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online]. 2001.

[65] Upulee Kanewala and James M. Bieman. “Testing scientific software: A systematic literature review”. In: *Information and Software Technology* 56.10 (2014), pp. 1219–1232. arXiv: `arXiv:1804.01954v1`.

[66] George Karypis and Vipin Kumar. “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”. In: *SIAM Journal on Scientific Computing* 20.1 (Jan. 1998), pp. 359–392.

[67] Hermann Kopetz and Wilhelm Ochsenreiter. “Clock Synchronization in Distributed Real-Time Systems”. In: *IEEE Transactions on Computers* C-36.8 (1987), pp. 933–940.

[68] Michael Kornhaas, Michael Schäfer, and Dörte C. Sternel. “Efficient numerical simulation of aeroacoustics for low Mach number flows interacting with structures”. In: *Computational Mechanics* 55.6 (2015), pp. 1143–1154.

[69] Q. T. Le Gia and Holger Wendland. “Data compression on the sphere using multiscale radial basis function approximation”. In: *Advances in Computational Mathematics* 40.4 (Aug. 2014), pp. 923–943.

[70] Florian Lindner et al. “A comparison of various quasi-newton schemes for partitioned fluid-structure interaction”. In: *A comparison of various quasi-newton schemes for partitioned fluid-structure interaction*. 2015, pp. 1–12.

[71] Yang Luo et al. “A fluid-structure interaction study on a passively deformed fish fin”. In: *Proceedings of the ASME 2019 38th International Conference on Ocean, Offshore and Arctic Engineering*. June. Glasgow, Scotland, 2019.

[72] Fachgebiet Numerische Berechnungsverfahren im Maschinenbau. *FASTEST Manual*. 2005.

[73] Vladimir Maz’ya and Gunther Schmidt. “On approximate approximations using Gaussian kernels”. In: *IMA Journal of Numerical Analysis* 16.1 (1996), pp. 13–29. arXiv: `arXiv:1011.1669v3`.

[74] Steve McConnell. *Code Complete: A Practical Handbook of Software Construction*. 2nd ed. Best Practices for Developers. Redmond, WA: Microsoft Press, 2004.

[75] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 51–56.

[76] Miriam Mehl et al. “The Scalability-Efficiency/Maintainability-Portability Trade-off in Simulation Software Engineering: Examples and a Preliminary Systematic Literature Review”. In: *2016 Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*. 2016, pp. 26–34.

[77] Nick Radcliffe (Cray Inc.) *MPI Dynamic Process Management MPI*. Tech. rep. 2018, pp. 1–10.

[78] K. C. Park and Carlos A. Felippa. “A variational principle for the formulation of partitioned structural systems”. In: *International Journal for Numerical Methods in Engineering* 47.1-3 (Jan. 2000), pp. 395–418.

[79] Rodrigo B. Platte. “How fast do radial basis function interpolants of analytic functions converge?” In: *IMA Journal of Numerical Analysis* 31.4 (2011), pp. 1578–1597. arXiv: `1409.2923`.

[80] Rodrigo B. Platte and Tobin A. Driscoll. “Polynomials and Potential Theory for Gaussian Radial Basis Function Interpolation”. In: *SIAM Journal on Numerical Analysis* 43.2 (2005), pp. 750–766.

[81] J. Postel. *User Datagram Protocol*. RFC 768. Internet Engineering Task Force, Aug. 1980.

[82] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*. New York; Heidelberg [u.a.]: Springer, 2000.

[83] Michel Raynal. “Parallel Computing vs. Distributed Computing: A Great Confusion? (Position Paper)”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Ed. by Sascha Hunold et al. Vol. 9523. Cham: Springer International Publishing, 2015, pp. 41–53.

[84] Leibniz Rechenzentrum. *LRZ: Using GPFS*. 2019. URL: `https://www.lrz.de/services/compute/supermuc/filesystems/gpfs-usage/#donot` (visited on 04/09/2019).

[85] Shmuel Rippa. “An algorithm for selecting a good value for the parameter c in radial basis function interpolation”. In: *Advances in Computational Mathematics* 11 (1999), pp. 193–210.

[86] H. H. Rosenbrock. “An Automatic Method for Finding the Greatest or Least Value of a Function”. In: *The Computer Journal* 3.3 (1960), pp. 175–184. eprint: `/oup/backfile/content_public/journal/comjnl/3/3/10.1093/comjnl/3.3.175/2/030175.pdf`.

[87] Michael R. Ross et al. “Treatment of acoustic fluid-structure interaction by localized Lagrange multipliers: Formulation”. In: *Computer Methods in Applied Mechanics and Engineering* 197.33-40 (2008), pp. 3057–3079.

[88] Youcef Saad and Martin H. Schultz. “GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems”. In: *SIAM Journal on Scientific and Statistical Computing* 7.3 (1986), pp. 856–869. arXiv: `fld.1 [DOI: 10.1002]`.

[89] Marta Camps Santasmasas et al. “Dual Navier-Stokes / Lattice-Boltzmann Method for Urban Wind Flow”. In: *12th International ERCOFTAC Symposium on Engineering Turbulence Modelling and Measurements (ETMM12)*. Montpellier, France, 2018.

[90] Robert Schaback. “Error estimates and condition numbers for radial basis function interpolation”. In: *Advances in Computational Mathematics* 3.3 (Apr. 1995), pp. 251–264.

[91] Robert Schaback. “Native Hilbert Spaces for Radial Basis Functions I”. In: *New Developments in Approximation Theory*. Ed. by Manfred W. Müller et al. Basel: Birkhäuser Basel, 1999, pp. 255–282.

[92] Klaudius Scheufele. “Coupling Schemes and Inexact Newton for Multi-Physics and Coupled Optimization Problems”. PhD Thesis. University of Stuttgart, 2019.

[93] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit*. 4th ed. Clifton Park, N.Y: Kitware, 2006.

[94] Judith Segal. “Models of scientific software development”. In: *SECSE 08, First International Workshop on Software Engineering in Computational Science and Engineering*. Leipzig, Germany, 2008.

[95] C.E. Shannon. “Communication in the Presence of Noise”. In: *Proceedings of the IRE* 37.1 (Jan. 1949), pp. 10–21.

[96] S. Sharma, D. de Santis, and A. Shams. “An advanced URANS solver to predict turbulence induced vibrations in nuclear reactor applications”. In: *Computational Fluid Dynamics (CFD) in Nuclear Reactor Safety*. 2018, pp. 1–13.

[97] David Sommer. “Stable Radial Basis Function Interpolation for Multi-Physics Simulation Applications”. Bachelor thesis. University of Stuttgart, 2018.

[98] V. Strouhal. “Über eine besondere Art der Tonerregung”. In: *Annalen der Physik* 241.10 (1878), pp. 216–251. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/andp.18782411005`.

[99] Hiroyuki Takizawa et al. “Xevolver: An XML-based code translation framework for supporting HPC application migration”. In: *2014 21st International Conference on High Performance Computing, HiPC 2014*. IEEE, 2014, pp. 1–11.

[100] Robert L. Taylor and Sanjay Govindjee. *FEAP - - A Finite Element Analysis Program*. English. Version 8.5. University of California at Berkeley.

[101] Claudio E. Torres and L. A. Barba. “Fast radial basis function interpolation with Gaussians by localization and iteration”. In: *Journal of Computational Physics* 228.14 (Aug. 2009), pp. 4976–4999.

[102] Joel Aaron Tropp. “Topics in Sparse Approximation”. PhD thesis. University of Texas at Austin, 2004.

[103] Stefan Turek and Jaroslav Hron. “Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow”. In: *Fluid-Structure Interaction* 53 (2006), pp. 371–385.

[104] Greg Turk. *The PLY Polygon File Format*. `https://web.archive.org/web/20161204152348/http://www.dcs.ed.ac.uk/teaching/cs4/www/graphics/Web/ply.html`. 1994.

[105] Benjamin Uekermann. “Partitioned Fluid-Structure Interaction on Massively Parallel Systems”. PhD Thesis. Technical University of Munich, 2016.

[106] H. A. van der Vorst. “Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems”. In: *SIAM Journal on Scientific and Statistical Computing* 13.2 (Mar. 1992), pp. 631–644.

[107] Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. “The NumPy Array: A Structure for Efficient Numerical Computation”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30. eprint: `https://aip.scitation.org/doi/pdf/10.1109/MCSE.2011.37`.

[108] Tianyang Wang et al. “Assessment and practical application of mapping algorithms for beam elements in computational FSI”. In: *European Journal of Computational Mechanics* 25.5 (2016), pp. 417–445.

[109] H. G. Weller et al. “A tensorial approach to computational continuum mechanics using object-oriented techniques”. In: *Computers in Physics* 12.6 (1998), p. 620.

[110] Holger Wendland. “Multiscale Radial Basis Functions”. In: *Frames and Other Bases in Abstract and Function Spaces*. Ed. by I. Pesenson et al. Vol. 1. Birkhäuser, Cham, 2017, pp. 265–299.

[111] Barbara I. Wohlmuth. “A Mortar Finite Element Method Using Dual Spaces for the Lagrange Multiplier”. In: *SIAM Journal on Numerical Analysis* 38.3 (2000), pp. 989–1012.

[112] Qi Zhang, Yangzhang Zhao, and Jeremy Levesley. “Adaptive radial basis function interpolation using an error indicator”. In: *Numerical Algorithms* (Jan. 2017).

[113] Jens Zudrop et al. “A fully distributed CFD framework for massively parallel systems”. In: *Cray User Group Conference*. Stuttgart: Cray User Group, 2012.

Declaration of Authorship

I hereby declare that this thesis titled

Data Transfer in Partitioned Multi-Physics Simulations:
Interpolation & Communication

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

Stuttgart, April 30, 2019

Publications

Publications

- *F. Lindner, M. Mehl, B. Uekermann:* **Radial Basis Function Interpolation for Black-Box Multi-Physics Simulations**
  International Center for Numerical Methods in Engineering (CIMNE), Proceedings of the VII International Conference on Coupled Problems in Science and Engineering, 2017.

- *H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann:* **preCICE – A Fully Parallel Library for Multi-Physics Surface Coupling**
  Computers & Fluids, Elsevier, 2016.

- *H.-J. Bungartz, F. Lindner, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann:* **Partitioned FluidStructureAcoustics Interaction on Distributed Data: Coupling via preCICE**
  H.-J. Bungartz, P. Neumann, W.E. Nagel: Software for Exascale Computing – SPPEXA 2013-2015, Springer International Publishing, 2016.

- *D. Pflüger, M. Mehl, J. Valentin, F. Lindner, D. Pfander, S. Wagner, D. Graziotin, Y. Wang:* **The Scalability-Efficiency/Maintainability-Portability Trade-Off in Simulation Software Engineering: Examples and a Preliminary Systematic Literature Review**
  Proceedings of 2016 Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SE-HPCCSE 2016), held in conjunction with SC16, Salt Lake City, Utah.

- *D. Blom, F. Lindner, M. Mehl, K. Scheufele, A. van Zuijlen:* **A Review on Fast Quasi-Newton and Accelerated Fixed Point Iterations for Partitioned Fluid-Structure Interaction Simulation**

Advances in Computational Fluid-Structure Interaction, Springer International Publishing, 2015.

- *H.-J. Bungartz, H. Klimach, V. Krupp, F. Lindner, M. Mehl, S. Roller, B. Uekermann:* **Fluid-Acoustics Interaction on Massively Parallel Systems**
  M. Mehl, M. Bischoff, Manfred, M. Schäfer: Recent Trends in Computational Engineering, Berlin, Heidelberg, New York: Springer, 2015.

- *F. Lindner, M. Mehl, B. Uekermann:* **preCICE - A versatile partitioned coupling library**
  Poster, SIAM CSE 2015, Salt Lake City.

- *H.J. Bungartz, F. Lindner, M. Mehl, B. Uekermann:* **A plug-and-play coupling approach for parallel multi-field simulations**
  Computational Mechanics. Vol. 55(6), Berlin, Heidelberg, New York: Springer, 2015.

- *F. Lindner, M. Mehl, K. Scheufele, B. Uekermann:* **A Comparison of various Quasi-Newton Schemes for Partitioned Fluid-Structure Interaction**
  Coupled Problems, 2015.