Ingenieurfakultät Bau Geo Umwelt
Lehrstuhl für Computation in Engineering

Towards scalable finite cell computations on massively parallel systems

John Njuguna Jomo, M.Sc.

Vollständiger Abdruck der von der Ingenieurfakultät Bau Geo Umwelt der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:		Prof. Dr.-Ing. André Borrmann
Prüfer der Dissertation:	1.	Prof. Dr. rer. nat. Ernst Rank
	2.	Assoc. Prof. Dr. ir. Clemens V. Verhoosel
	3.	Priv.-Doz. Dr. rer. nat. habil. Ralf-Peter Mundani

Die Dissertation wurde am 11.11.2020 bei der Technischen Universität München eingereicht und durch die Ingenieurfakultät Bau Geo Umwelt am 16.03.2021 angenommen.

Zusammenfassung

Die numerische Analyse von Körpern mit topologisch und geometrisch komplizierter Struktur stellt eine Herausforderung in der Ingenieurspraxis dar. Wenn herkömmliche Finite Elemente Methoden angewendet werden, kann die Erstellung eines Körper-angepassten Netzes äußerst umständlich sein und einen großen Teil der von Ingenieuren aufgewendeten Zeit ausmachen. Es besteht daher ein wachsender Bedarf an robusten Diskretisierungstechniken für komplexe Geometrien. Diese Techniken sollen nicht nur eine einfache Handhabung aufweisen, sondern auch auf große Probleme von technischer Relevanz angewendet werden können, an denen mehrere Millionen oder sogar Milliarden von Unbekannten beteiligt sind. Fiktive Gebietsmethoden bieten einen Rahmen für die Analyse von Körpern mit komplexen Geometrien. Sie zielen nicht darauf ab, den Rand einer Geometrie konform abzubilden, sondern platzieren den Körper in ein Einbettungsnetz mit einer einfachen Struktur. Die Verwendung einer Einbettungsdiskretisierung reduziert den Aufwand für die Netzgenerierung erheblich.

Die weit verbreitete Verwendung von fiktiven Gebietsmethoden zur Berechnung von großen Problemen war eine lange Zeit durch zwei Hauptgründe begrenzt. Erstens weisen eingebettete Methoden eine schlechte Konditionierung auf. Dieses Phänomen ist auf Elemente zurückzuführen, welche von der Oberfläche der eingebetteten Geometrie geschnitten werden. Die schlechte Konditionierung hat die Verwendung iterativer Gleichungslöser, welche für das Lösen großer linearer Systeme gut geeignet sind, erschwert. Bis vor wenigen Jahren haben Wissenschaftler und Forscher hauptsächlich auf die Verwendung von direkten Lösern zurückgegriffen, die für große Gleichungssysteme nur begrenzt anwendbar sind. Zweitens gibt es nur wenige numerische Codes, mit denen man eingebettete Berechnungen mit mehreren Millionen Freiheitsgraden durchführen kann. Solche Codes erfordern ein ordnungsgemäßes Design, um eine effiziente Ausnutzung der Rechenressourcen auf massiv parallelen Systemen zu erzielen.

In dieser Arbeit werden Algorithmen und Datenstrukturen vorgestellt, die große Berechnungen mittels der Finite-Zellen-Methode ermöglichen. Darin wird gezeigt, wie angepasste Vorkonditionierungsstrategien angewendet werden können, um die Konditionierungsprobleme zu verbessern, welche in Zusammenhang mit der Finite-Zellen-Methode auf gleichmäßige Gitter und mehrstufige, *hp*-verfeinerte Netze auftreten. Es wird ein massiv-paralleles Framework entwickelt, welches Berechnungen mit bis zu 98000 Prozessoren erlaubt. Die in dieser Arbeit entwickelten Methoden finden Anwendung in der additiven Fertigung von metallischen Bauteilen. Erstens helfen großangelegte parallele Simulationen bei der Materialcharakterisierung von additiv gefertigten Teilen. Der zweite Anwendungsbereich ist die Simulation von Fertigungsprozessen. Hierzu werden speziell entwickelte parallele Verfeinerungsstrategien verwendet, welche die numerische Analyse von additiven Herstellungsverfahren ermöglichen.

Abstract

Numerical analysis of bodies with complex shapes poses a challenge in engineering practice. When standard finite element methods are applied, the process of generating a body-fitting mesh can be exceedingly cumbersome and may make up the bulk of time spent by practitioners. There is, therefore, a growing need for robust discretizational techniques that not only allow an easy treatment of complex geometries, but that can be also applied to large-scale problems of engineering relevance involving multiple millions or even billions of unknowns. Immersed finite element methods provide a framework for the analysis of bodies with complex geometries. They do not aim to resolve the boundary of a complex geometry, but rather place the body in an embedding mesh with a simple structure. The use of an embedding discretization significantly reduces the effort related to mesh generation.

The widespread use of immersed finite element methods for the computation of large-scale problems has for a long time been limited by two main reasons. First, immersed methods are proven to ill-conditioning due to the elements that are intersected by the boundary of the embedded geometry. This made the use of iterative solvers, that are well-suited for the solution of large linear systems, impractical in many cases since methods for the systematic treatment of ill-conditioning where scarce. Scientists and researchers would mainly resort to the use of direct solvers, which have limited applicability for large problem sizes. Secondly, there are only a few numerical codes, capable of performing immersed computations with multiple millions of degrees of freedom. Such codes require a proper design in order to run efficiently on massively parallel systems.

This thesis presents algorithms and data structures needed for large-scale immersed computations using the finite cell method. It demonstrates how dedicated preconditioning strategies can be used to ameliorate conditioning problems associated with the finite cell method on uniform meshes and multi-level *hp*-refined grids. A massively parallel framework is developed that can perform immersed computations with up to 98000 processors. The methods presented in this work are applied to second-order elliptic problems as well as in two areas in the field of metal additive manufacturing. First, large-scale parallel simulations aid in the virtual material characterization of additively manufactured parts. The second application area is the simulation of the manufacturing process itself. Specially designed parallel refinement strategies are used that allow the numerical analysis of the fabrication procedure.

Preface

This thesis was created during my time as an employee and PhD researcher at the Chair for Computation in Engineering at the Technical University of Munich from December 2014 to June 2020. I would like to take this opportunity to express my gratitude to all the people and institutions that contributed to the success of this work.

First of all, I thank my doctoral supervisor, Prof. Ernst Rank, for his continual support and guidance throughout my thesis. He granted me the freedom to expand my scientific knowledge by exploring different research topics. I appreciate his valuable advice, the opportunities he offered me to participate in numerous courses and scientific events and his ability to bring people together, which resulted in fruitful collaborations.

I also thank Dr. Clemens V. Verhoosel for agreeing to be part of my doctoral committee. I got the opportunity to visit him in 2017 and to spend time with his research group in Eindhoven. This experience was an important milestone in my doctoral research project and I am grateful for the collaboration between Munich and Eindhoven.

Furthermore, I thank PD Dr. Ralf-Peter Mundani for being part of my PhD committee. Dr. Mundani has been of great help over the years in providing guidance and advice on supercomputing questions. He was also involved in several successful applications for computing resources and funding.

I also thank Prof. Dr.Ing. Andŕe Borrmann for serving as the chairman during my thesis examination.

I would like to extend my gratitude to Dr. Stefan Kollmannsberger, the team leader of the Simulations in Applied Mechanics group, of which I was a proud member. Thank you for providing a conducive work environment and a friendly atmosphere, that made work at the chair a great joy. I enjoyed our long discussions and brainstorming sessions in and out of the office and the unrelenting support you provided at every stage of the doctoral thesis.

One pivotal experience during my time as a doctoral candidate was my research stay at the Technical University of Eindhoven from April 2017 to May 2017. I would like to thank Prof. Harald Van Brummelen and Dr. Clemens Verhoosel and Dr. Frits de Prenter for hosting me during this time. I am grateful for the many fruitful discussions and invaluable insight you provided on preconditioning techniques for immersed methods. In particular, I would like to extend my deepest gratitude to Frits for his friendship and unwavering support even after my stay in Eindhoven. Thank you for sharing your knowledge with me. I truly appreciate the numerous Skype sessions, constructive critique and the encouraging words, which were integral to the completion of this thesis.

My international research stay at the University of Pavia from September 2019 to October 2019 was another important experience during my time as a PhD researcher. I want to express my sincere thanks to my hosts Prof. Ferdinando Auricchio, Prof. Alessandro Reali and Dr. Massimo Carraturo. I appreciate the time spent in Pavia, the helpful contributions and suggestions you provided and your in-depth knowledge of numerical methods and material modeling. I especially want to thank Massimo for organizing all the paperwork for my stay, for the many discussions over Skype and the fruitful collaboration on immersed layerwise simulations for metal additive manufacturing processes.

Furthermore, I thank all my colleagues, past and present, who made working at the Chair for Computation in Engineering an unforgettable experience. I will cherish the time we spent

together, the lively discussions and lunch sessions and the experiences we shared during and outside work. Many thanks to (in alphabetical order) Tino Bog, Davide D'Angella, Mohamed Elhaddad, Christoph Ertl, Bobby Ginting, Lisa Hug, Philipp Kopp, Nina Korshunova, László Kudela, Alexander Paolini, Nevena Perovic, Jing Rao, Robert Seidl, Benjamin Wassermann and Nils Zander. I also want to thank Hanne Cornils for handling all administrative matters with great care and efficiency. Special thanks go to Ali Özcan, with whom I shared the pleasure of sharing an office for close to four years. I am grateful for the many laughs, discussions and great time we had together.

I want to also thank the various institutions that supported the work and activities done during my doctoral thesis. First and foremost, I want to thank the International Graduate School of Science and Engineering (IGSSE) for their financial support and help in developing my scientific and personal skills. IGSSE enabled me to attend several international conferences and workshops and spend a total of three months at internationally renowned universities. I also want to acknowledge the support of the Kompetenznetzwerk für wissenschaftliches Höchstleistungsrechnen in Bayern (KONWIHR) and the Leibniz Supercomputing Center (LRZ) who provided the financial and technical support needed to develop efficient code for numerical simulations with the finite cell method.

Furthermore, I want to express my sincerest gratitude to my family and friends for their unrelenting support. I thank my parents for always encouraging me to strive for academic excellence and their sacrifice in helping me achieve my goals.

I thank my wife Frauke, for supporting me in every possible way. The completion of this thesis would not have been possible without your steadfast love and sacrifice. Last but not least, I want to thank God for giving me the strength, patience and opportunity to undertake this doctoral project and see it through to its completion.

John Njuguna Jomo
Munich, July 2020

Contents

Chapter 1

Introduction

1.1 Motivation

Numerical methods have become indispensable in modern engineering practice. The finite element method (FEM) in particular has emerged as the method of choice for numerous problems in mechanical and civil engineering. It is applied in problems where the underlying physical phenomena can be described through partial differential equations (PDEs). FEM is used, for example, in the design of optimized components, to analyze the strength and durability of load-bearing structures, to predict the onset of material failure and to gain insight on the influence of process parameters in manufacturing techniques such as die casting, metal forming and additive manufacturing.

The widespread use of FEM in the engineering community has been largely driven by advances in the field of computing. Today, computers help perform many engineering tasks that used to be executed manually. Their impact can be seen in several modern disciplines such as computer-aided engineering (CAE), computer-aided design (CAD) and computer-aided manufacturing (CAM).

The last few decades have seen an upward trend in the complexity of finite element simulations. Nowadays, multiphysics and multiscale simulations allow a more accurate description of real-life systems compared to FE analyses in the past, by coupling different physical fields and incorporating information from varying spatial and/or temporal scales. Parallel computing is now widely used in modern finite element codes, enabling users to exploit the computational power, in terms of speed and space, of many-core and distributed memory machines. Furthermore, the geometrical models used in FE simulations have also become increasingly sophisticated. Modern simulation frameworks have to handle different types of geometrical input data such as CAD files, voxel-based data and implicit models, and also be able to perform analyses on bodies with a complex shape and morphology.

Although tremendous progress has been made towards more realistic and efficient numerical simulations, the development of finite element codes capable of handling complicated physics and complex geometries still has many challenges that need to be addressed and opportunities that remain undiscovered. The three sources of complexity, mentioned in the previous paragraph, continue to be the central theme of many research topics to date. Several alternatives to and variants/extensions of the finite element method have been developed over the years to cater to the growing needs in industry and provide novel solutions to existing problems.

Notable examples include isogeometric analysis (IGA) that strives for a seamless integration of CAD geometries in numerical simulations and the extended finite element method (XFEM) that provides a mechanism for treating discontinuities due to cracks and material interfaces in finite element simulations.

Unfitted, immersed and fictitious domain finite element methods are innovative techniques that aim to provide scientists and engineers with a robust and versatile FE framework for handling complex geometries. In a practitioner's day-to-day work, a lot of time and effort is spent in the generation of body-fitting meshes that are typically required in standard finite element approaches. Immersed methods apply a methodology that significantly simplifies the task of mesh generation for complex geometries and allows a straight-forward incorporation of different geometric models and input data in numerical simulations. They employ a discretization that does not need to capture the boundaries of a complex body, but rather embed the body in a surrounding domain with a simple structure. This procedure results in elements that lie completely inside the original body and *cut elements* that are intersected by the body's boundary.

One immersed finite element method that has been gaining traction over the last few years is the finite cell method (FCM). It combines the flexibility of fictitious domain methods with the approximation qualities of high-order finite elements, yielding a powerful scheme for performing numerical analyses on complex geometries. This method has been applied to various problems in fluid and solid mechanics, providing quality results with fewer degrees of freedom than standard boundary-conforming approaches. FCM is often used in conjunction with the multilevel *hp*-refinement, a novel *hp*-method that allows the resolution of fine-scale features in critical regions of the computational mesh.

A major difficulty that faces many immersed methods, including the finite cell method, is the extension of these approaches to allow for efficient, large-scale computations for problems of engineering relevance. Such computations require well-designed parallel codes that allow the scalable generation and storage of mesh elements as well as efficient algorithms that can leverage the capabilities of modern computing clusters. It was only until recently that the inherent conditioning problems due to the cut elements in immersed methods were systematically analyzed and numerically verified. In fact, the development of stabilization and preconditioning techniques for immersed methods is a vibrant research field, with numerous publications and ongoing research projects focused on this subject. These developments have not only begun to open the door for the use of iterative solvers in the immersed community but have also created a demand for efficient parallel immersed frameworks that can be applied to a variety of problem classes. To date, only a few codes are capable of performing large-scale immersed computations with multiple millions and even billions of unknowns.

1.2 Objectives

The aim of this thesis is the development of efficient algorithms and data structures for massively parallel computations using the finite cell method and multi-level *hp*-refinement. To accomplish this goal it is required to innovate various aspects of the FCM framework, most notably *i)* the design of scalable algorithms and data structures for the management of parallel finite cell grids, *ii)* the development of robust and efficient iterative solution techniques for finite cell discretizations and *iii)* the application of the developed methods in problems of

engineering relevance.

A key feature of this work is the light-weight distributed data structures based on an adaptive Cartesian grid that is utilized for mesh creation and manipulation in a parallel setting. Although the use of adaptive Cartesian grids is not new, the approach presented in this work is specially adapted to the finite cell method and multi-level *hp*-refinement. Another cornerstone of this thesis is the use of dedicated preconditioning techniques based on the additive Schwarz lemma to resolve the conditioning issues of finite cell systems. In this thesis, the pioneering work of de Prenter et al. [2017] on preconditioners for uniform immersed discretizations is extended to finite cell systems involving multi-level *hp*-refinement and further improved by the use of hierarchical multigrid solution techniques. These additions provide engineers with new possibilities for the iterative solution of large finite cell systems.

This thesis also illustrates how well-designed parallel data structures and iterative solution strategies can be used hand-in-hand to solve engineering problems involving complex geometries. Examples of such problems are shown from the field of linear elasticity and metal additive manufacturing processes.

1.3 Outline

The thesis at hand is organized into seven chapters. **Chapter 2** introduces the concepts, terminology and notation needed to understand subsequent parts of this work. It reviews the fundamental principles of the finite element method and its *h*-, *p*- and *hp*-extensions. The chapter presents the core concept behind immersed methods and highlights the influence of cut elements on numerical quadrature, strong boundary condition imposition and the conditioning of immersed systems. The chapter concludes with a brief review of prevalent immersed methods.

A summary of the finite cell method and the multi-level *hp*-method is given in **Chapter 3**. These discretization techniques form the basis of this work and the nomenclature used in connection with these approaches is presented. This chapter also introduces the numerical framework, `AdhoC++`, that is used in this thesis.

Chapter 4 begins by revisiting the fundamentals of parallel computing. It mentions the essential ingredients for scalable finite elements in general, i.e. scalable mesh management, robust and efficient iterative solvers and scalable post-processing and provides a review of parallel frameworks for high-order finite element methods. The chapter also presents two parallelization strategies for FCM and multi-level *hp*-refinement. The first scheme is based on replicated data structures and is suitable for small computing clusters. The second approach is based on a fully distributed adaptive Cartesian grid that allows large-scale simulations on massively parallel systems.

The development of robust iterative solution schemes for finite cell problems involving multi-level *hp*-refinement is handled in **Chapter 5**. The chapter recapitulates the underlying cause of ill-conditioning in FCM and presents theoretical and implementational aspects of additive Schwarz preconditioning strategies for uniform finite cell meshes and multi-level *hp*-refined finite cell grids. A multi-grid framework that takes advantage of the hierarchical structure in the finite cell method and multi-level *hp*-refinement is presented at the end of this chapter. A series of numerical examples show the effectiveness of the proposed solution strategies for large-scale finite cell analysis of problems of industrial relevance.

Chapter 6 illustrates how the methods development in this thesis can be applied in the context of metal additive manufacturing processes. Two application areas are considered, the simulation of additively manufacturing products and the simulation of the fabrication process itself. The first part of the chapter shows how large-scale finite cell computations can be used in the virtual material characterization of additively manufactured foams while the use of immersed technology for high-fidelity and layerwise simulations of the selective laser melting processes is shown in the second part of the chapter.

This thesis concludes in **Chapter 7** with a summary and outlook.

Chapter 2

The finite element method

The chapter at hand lays the foundation for understanding the algorithms and methods presented in this work. It briefly introduces the concepts, terminology and notation used in this thesis. A comprehensive overview of the finite element method (FEM) can be found in different works such as [Bathe, 2007; Ciarlet, 2002; Hughes, 2000; Larson and Bengzon, 2013; Szabó and Babuška, 1991]. FEM is a numerical method used to find approximate solutions of partial differential equations (PDEs) posed on a domain of computation. It relies on the transformation of the differential or strong form of a PDE, that has to be fulfilled in every point in a body Ω, into an expression termed the weak or integral form that only needs to be fulfilled in an integral sense. This transformation is usually carried out using variational principles that are applied to the differential equations and their respective boundary conditions. Boundary segments along which the value of the solution is prescribed are known as Dirichlet boundaries, denoted by $\partial\Omega_D$, while those along which the derivatives are specified are called Neumann boundaries, which are denoted by $\partial\Omega_N$.

The continuous weak form of second-order PDEs, such as those arising in linear problems in heat exchange and elasticity, can be written as

$$\text{Find } u \in \mathcal{V} \text{ that satisfies } a(u,v) = f(v) \;\; \forall\, v \in \mathcal{V}_0, \tag{2.1}$$

where the term u denotes the solution of the weak form, v an arbitrary test function, $a(\cdot,\cdot)$ a bilinear symmetric operator and $f(\cdot)$ a linear operator. The terms $\mathcal{V}$ and $\mathcal{V}_0$ both represent a set of functions, in which each function contained in the set and its first derivative is square-integrable. These sets differ when a non-zero function denoted by g_D is prescribed on the Dirichlet boundary. The function $u \in \mathcal{V}$ assumes a value of g_D on $\partial\Omega_D$ while $v \in \mathcal{V}_0$ is equal to zero on $\partial\Omega_D$. The following section elaborates on the steps needed to transform the continuous weak form into a set of algebraic equations using the finite element method.

2.1 The Galerkin finite element method

The Galerkin finite element method transforms the continuous infinite-dimensional problem in (2.1) into a discrete finite-dimensional problem by introducing a finite-dimensional subspace $\mathcal{V}_h \subset \mathcal{V}$ in which a solution u_h that approximates u is sought. This allows (2.1) to be reformulated as

$$\text{Find } u_h \in \mathcal{V}_h \text{ that satisfies } a(u_h,v_h) = f(v_h) \;\; \forall\, v_h \in \mathcal{V}_{0,h}. \tag{2.2}$$

2.1.1 Finite element discretization

In order to explain the core principle behind the finite element method, a body Ω with Dirichlet boundaries $\partial\Omega_D$ and Neumann boundaries $\partial\Omega_N$ is considered, where $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$ and $\partial\Omega_D \cap \partial\Omega_N = \emptyset$. The subspace $\mathcal{V}_h$ is constructed in two steps. First, the problem domain Ω is partitioned into a set of nonoverlapping subdomains called *finite elements* in a process known as *discretization*. The symbol K is used to represent a single element while the mesh, denoted by $\mathcal{T}_h$, represents the union of all elements.

Ciarlet [2002] postulates requirements on the geometrical form of a finite element, stating that K must be a closed subset of Ω with a non-empty interior and Lipschitz-continuous boundary ∂K. It is common to define the geometry of an element $K \in \mathbb{R}^d$ using a d-dimensional polytope — such as a triangle or quadrilateral in two dimensions or using a tetrahedron or hexahedron in three dimensions — but elements with arbitrarily curved boundaries can also be used. The element boundary ∂K is made up of m-dimensional manifolds with $0 \leq m \leq d-1$. These topological entities can be classified according to their dimensionality as *nodes* for $m = 0$, *edges* for $m = 1$ and *faces* for $m = 2$.

A finite element mesh $\mathcal{T}_h$ is associated with a mesh parameter h that represents an element's characteristic length. A large value of h corresponds to a mesh with large elements while a small value of h is associated with small elements. A mesh with few large elements is usually termed as "coarse" as opposed to a "fine" mesh, which comprises small elements.

Once the mesh $\mathcal{T}_h$ has been set up, the second step in the construction of $\mathcal{V}_h$ can be performed. This involves associating each element with an element space $\mathcal{P}_K \subset C^s(K)$, a finite-dimensional C^s-continuous vector space defined on K. It is common to choose $\mathcal{P}_K$ as a space of mapped polynomials since they can be easily evaluated. Spaces based on different functions can also be chosen, e.g. non-rational functions [Cottrell et al., 2009], but this is not considered in this work. In order to introduce the terminology used in this manuscript, a polynomial element space $\mathcal{P}_K$ of polynomial degree p and dimensionality n_p is considered. This space is associated with a set of n_p linear independent element basis functions $\mathbf{N}_K$ defined on K and a set of n_p coefficients $\mathbf{u}_K$ defined as

$$\mathbf{N}_K = \{\varphi_{i,K}\}_{i=1}^{n_p} \quad , \quad \mathbf{u}_K = \{\mathrm{u}_{i,K}\}_{i=1}^{n_p}\,, \tag{2.3}$$

where $\varphi_{i,K}$ represents the i^{th} polynomial element basis function and $\mathrm{u}_{i,K} \in \mathbb{R}$ the i^{th} element coefficient. Each $\mathrm{u}_{i,K}$ is linked to a specific $\varphi_{i,K}$ allowing an arbitrary polynomial function $u_K \in \mathcal{P}_K$ to be represented as a linear combination of element basis functions and their respective coefficients as per the formula

$$u_K = \sum_{i=1}^{n_p} \mathbf{N}_K \; \mathrm{u}_{i,K}. \tag{2.4}$$

The element spaces in the mesh collectively provide an elementwise definition of the subspace $\mathcal{V}_h$ since any given function $v_h \in \mathcal{V}_h$ can be represented through its restriction $v_h|_K = v_K$ on individual elements. The finite element space $\mathcal{V}_h$ can, therefore, be defined as

$$\mathcal{V}_h(\mathcal{T}_h) = \{v \in \mathcal{V} : v|_K \in \mathcal{P}_K, \quad \forall K \in \mathcal{T}_h\}. \tag{2.5}$$

In the finite element method, a differentiation is made between quantities that are defined on element-level (local-level) such as v_K and quantities defined on global-level such as v_h. Global

counterparts of $\mathbf{N}_K$ and $\mathbf{u}_K$ exist, namely the set of global basis functions $\mathbf{N}$ and the set of global coefficients, also called degrees of freedom (DOFs), denoted by $\mathbf{u}$, where

$$\mathbf{N} = \{\varphi_i\}_{i=1}^{n_{\text{DOFs}}} \quad , \quad \mathbf{u} = \{\mathrm{u}_i\}_{i=1}^{n_{\text{DOFs}}} . \tag{2.6}$$

In (2.6) n_{DOFs} denotes the number of degrees of freedom while φ_i and u_i represent the i^{th} global basis function and global coefficient, respectively. Note that the local coefficient $\mathrm{u}_{i,K}$ on an element K is equal to the restriction of its corresponding global coefficient on that element i.e. $\mathrm{u}_i|_K = \mathrm{u}_{i,K}$. In general, elementwise interpolation operators π_{ij}^K are needed to relate a global basis function φ_i to the local basis functions as shown in (2.7).

$$\varphi_i = \sum_j \pi_{ij}^K \, \varphi_{j,K} \quad \forall K \in \mathcal{T}_h. \tag{2.7}$$

Following equations (2.4) to (2.7), the function u_h can be written as

$$u_h = \sum_{i=0}^{n_{\text{DOFs}}} \varphi_i \, \mathrm{u}_i. \tag{2.8}$$

As already indicated in (2.1) and (2.2) a Bubnov-Galerkin framework is applied in this thesis, in which the same space $\mathcal{V}_h$ is chosen for both u_h and v_h. Consequently, the weak form can be transformed into the expression

$$\sum_{j=1}^{n_{\text{DOFs}}} a(\varphi_i, \varphi_j) \, \mathrm{u}_j = f(\varphi_i), \quad \forall i \in \{1, \ldots, n_{\text{DOFs}}\}. \tag{2.9}$$

The discrete weak form in (2.9) can be rewritten in matrix-vector notation as

$$\mathbf{A}\mathbf{u} = \mathbf{F} \quad \text{with} \quad A_{ij} = a(\varphi_i, \varphi_j) \quad \text{and} \quad F_i = f(\varphi_i). \tag{2.10}$$

The matrix $\mathbf{A}$ is usually referred to as the *stiffness matrix*, while the right-hand side vector $\mathbf{F}$ is generally referred to as the *force vector*. It should be noted that the Neumann boundary conditions are incorporated in the operators $a(\cdot, \cdot)$ and $f(\cdot)$. Dirichlet boundary conditions can be applied in a systematic manner as discussed in several books on finite elements e.g. [Bathe, 2007; Hughes, 2000; Larson and Bengzon, 2013] using techniques generally referred to as strong imposition of Dirichlet boundary conditions. An alternative approach to these techniques is the weak imposition of Dirichlet boundary conditions. Weak imposition is applied in this thesis and will be presented in the context of immersed finite elements in Section 2.3.2.2. In subsequent sections, it will be assumed that the boundary conditions have already been incorporated in (2.10) such that $\mathbf{A}$ is invertible and a unique solution $\mathbf{u}$ exists.

2.1.2 Numerical quadrature and matrix assembly

It is common practice to evaluate the individual components of $\mathbf{A}$ and $\mathbf{F}$ on element-level and accumulate these values over all mesh elements in a process called *assembly*. The assembly procedure can be illustrated with the help of the assembly operator $\mathbf{A}$ as suggested in [Hughes, 2000] as

$$\mathbf{A} = \mathop{\mathbf{A}}_{e=1}^{n_{el}} \mathbf{A}^e, \tag{2.11a}$$

$$\mathbf{F} = \mathop{\mathbf{A}}_{e=1}^{n_{el}} \mathbf{F}^e, \tag{2.11b}$$

where $\mathbf{A}^e$ and $\mathbf{F}^e$ denote the element stiffness matrix and element force vector of the element K_e and n_{el} the number of elements in $\mathcal{T}_h$. The terms $\mathbf{A}^e$ and $\mathbf{F}^e$ can be computed as

$$A^e_{ij} = a(\varphi_{i,K_e}, \varphi_{j,K_e}), \tag{2.12a}$$
$$F^e_i = f(\varphi_{i,K_e}). \tag{2.12b}$$

When a concrete weak form such as that arising from a Poisson's problem is considered, then the element matrices in (2.12) can be rewritten in matrix-vector notation as

$$\mathbf{A}^e = \int_{K_e} (\nabla \mathbf{N}_K)^T (\nabla \mathbf{N}_K) \, \mathrm{d}\Omega, \tag{2.13a}$$
$$\mathbf{F}^e = \int_{K_e} \mathbf{N}_K^T f \, \mathrm{d}\Omega. \tag{2.13b}$$

In finite element codes, the evaluation of element vectors and matrices is usually simplified by introducing a *reference (standard) element* $\hat{K}$ with a simple shape with predefined element basis functions $\mathbf{N}_{\hat{K}} = \{\varphi_{i,\hat{K}}\}_{i=0}^{n_p}$. Instead of evaluating quantities, such as basis function values or their derivatives, directly on an arbitrarily shaped element $K \in \mathcal{T}_h$, one can first evaluate these quantities in the reference element $\hat{K}$ and map them into K. A mapping function $\boldsymbol{\Psi}_K$ can be defined for each element K that transforms a point $\boldsymbol{\xi}$ in the reference element coordinate system into a global point $\boldsymbol{x} \in K$

$$\boldsymbol{x} = \boldsymbol{\Psi}_K \, \boldsymbol{\xi}. \tag{2.14}$$

Moreover, a Jacobian matrix $\mathbf{J}_K$ is introduced for each element K that takes into account the change of variables from $\boldsymbol{x}$ to $\boldsymbol{\xi}$ when evaluating integrals and derivatives in $\hat{K}$ for a given K. This matrix can be computed as

$$\mathbf{J}_K(\xi) = \frac{\partial \boldsymbol{\Psi}_K}{\partial \boldsymbol{\xi}}. \tag{2.15}$$

Using the mapping function and the Jacobian matrix, it is possible to reformulate integrals over a given element K_e in terms of integrals over the reference element $\hat{K}$. Equation (2.13) formulated in terms of $\hat{K}$ reads

$$\mathbf{A}^e = \int_{\hat{K}} (\mathbf{J}_{K^e}^{-1} \nabla \mathbf{N}_{\hat{K}})^T (\mathbf{J}_{K^e}^{-1} \nabla \mathbf{N}_{\hat{K}}) \det(\mathbf{J}_{K^e}) \, \mathrm{d}\hat{\Omega}, \tag{2.16a}$$
$$\mathbf{F}^e = \int_{\hat{K}} \mathbf{N}_{\hat{K}}^T \, f(\boldsymbol{\Psi}_{K^e}) \det(\mathbf{J}_{K^e}) \, \mathrm{d}\hat{\Omega}, \tag{2.16b}$$

where $\hat{\Omega}$ denotes the domain of the reference element while $\mathbf{J}_{K^e}^{-1}$ and $\det(\mathbf{J}_{K^e})$ represent the inverse and the determinant of the Jacobian matrix of an element K^e, respectively. For quadrilateral elements, it is common to define a reference element $\hat{K}$ with a local coordinate system $\boldsymbol{\xi} = (\xi, \eta)$ where $\hat{\Omega} = [-1, 1] \times [-1, 1]$ and $\mathrm{d}\hat{\Omega} = \mathrm{d}\xi \, \mathrm{d}\eta$, see Figure 2.1.

Another common procedure in finite element codes is the numerical integration of element matrices. Numerical integration is more suitable for computer algorithms than analytical

evaluation of integrals since it relies on discrete operations that require less computational resources. It involves the evaluation of integrands at distinct points called *integrations (quadrature) points*, and the weighted summation of these values to yield the final result. The concept of numerical integration can be illustrated for a one-dimensional reference domain $\hat{\Omega} = [-1, 1]$ and $\boldsymbol{\xi} = \xi$ as

$$\int_{-1}^{1} f(\xi) \, \mathrm{d}\hat{\Omega} \approx \sum_{i=1}^{n_{\mathrm{GP}}} w_i \, f(\xi_i), \tag{2.17}$$

In (2.17), n_{GP} represents the number of integration points whereas $\xi_i \in \hat{\Omega}$ denotes the i^{th} integration point and w_i its corresponding weight. The set of n_{GP} pairs (ξ_i, w_i) used for numerical integration is usually referred to as a quadrature rule. Commonly used quadrature rules include the Gauss-Legendre quadrature [Lowan et al., 1942] and the Gauss-Lobatto quadrature [Chandrasekhar, 1960]. In this work, Gauss-Legendre quadrature is applied.

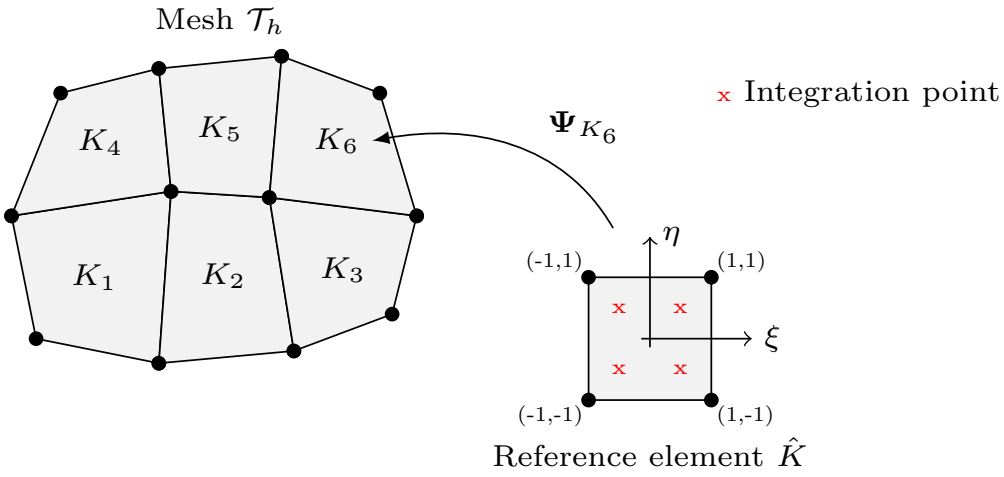


Figure 2.1: Illustration of a quadrilateral reference element $\hat{K}$ (adopted from [Zander, 2017]). Reference elements are used to simplify the numerical evaluation of element matrices.

2.1.3 Solution of the linear system

The discretized weak form in the finite element method can be transformed into an algebraic system of equations of the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.18}$$

where the matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is typically large and sparse. Various techniques can be employed to solve this system and they can be categorized as either *direct* or *iterative* solution methods. Direct solution methods are typically based on a factorization of the matrix $\mathbf{A}$ and yield an exact solution of the system. Common direct methods for large sparse systems include sparse implementations of Gaussian elimination, LU factorization and multifrontal decomposition. Iterative solution strategies provide an approximate solution of the system by repeatedly improving an initial guess of the solution until a specified tolerance is reached.

Direct solution approaches are typically robust, in the sense that their computational efficiency is not largely affected by the spectral properties of the matrix $\mathbf{A}$, given that this matrix is non-singular. Their computational cost i.e. the number of operations needed to solve a system and the amount of storage needed in the process, is primarily affected by the sparsity structure and the size of the matrix. Gaussian elimination generally has a computational cost in the order of $\mathcal{O}(n^3)$ in terms of floating-point operations (flops) and $\mathcal{O}(n^2)$ in terms of memory usage [Hackbusch, 1994]. Tailored sparse direct solvers can reduce the arithmetic complexity for certain matrix structures and thereby reduce the scaling rate of the computational cost. This can increase the size of solvable systems for direct solvers, but is still a bottleneck in large-scale finite element analyses with multiple millions or billions of unknowns. Furthermore, the parallelization of direct solvers is challenging, making them less suitable for parallel computing [Dongarra et al., 1998; Saad, 2003].

Iterative solution methods are more suitable for solving large systems than their direct counterparts since their the memory footprint and arithmetic cost scale better with the system size [Barrett et al., 1994]. This is attributed to the fact that they are mostly based on matrix-vector operations. Iterative methods can, moreover, be easily parallelized and in some cases allow a matrix-free implementation that further reduces their memory costs. The main drawback of iterative methods is that their effectiveness in solving a system is strongly dependent on the spectral properties of the system matrix $\mathbf{A}$. The convergence rate of an iterative methods denotes how the approximate solution in a given iteration $\mathbf{x}_i$ approaches the true solution. The convergence is often assessed by monitoring the development of the residual $\mathbf{r}_i$, defined as $\mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i$. A system with a well-clustered spectrum and a small spectral radius generally converges better than a system with a large unclustered spectrum. The spectrum $\lambda(\mathbf{A})$ of a matrix $\mathbf{A}$ denotes the set of eigenvalues of the matrix $\mathbf{A}$, i.e. $\lambda(\mathbf{A}) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. Systems with spectral properties that result in fast convergence of iterative solvers are referred to as well-conditioned, while those that lead to slow convergence are referred to as ill-conditioned. A commonly applied indicator of the conditioning of a system is the condition number $\kappa(\mathbf{A})$ defined as $\kappa(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$. This can be written for symmetric positive definite matrices as

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\| = \frac{\lambda_{\max}}{\lambda_{\min}} = \frac{\max_{\|\mathbf{v}\|=1} \mathbf{v}^T\mathbf{A}\mathbf{v}}{\min_{\|\mathbf{u}\|=1} \mathbf{u}^T\mathbf{A}\mathbf{u}}, \tag{2.19}$$

with $\lambda_{\max}$ and $\lambda_{\min}$ denoting the largest and smallest eigenvalues of $\mathbf{A}$ and the vectors $\mathbf{v}$ and $\mathbf{u}$ maximizing and minimizing the Rayleigh quotient.

The convergence of an ill-conditioned system can be improved by the application of a preconditioner $\mathbf{M}^{-1}$. Instead of solving the original ill-conditioned system, a modified system with better spectral properties is formed which has better convergence behavior. The most common approach for obtaining this modified system is by left preconditioning as shown in 2.20.

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}. \tag{2.20}$$

It is also possible to apply $\mathbf{M}^{-1}$ using right preconditioning

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{u} = \mathbf{b}, \text{ with } \mathbf{x} = \mathbf{M}^{-1}\mathbf{u}, \tag{2.21}$$

or as a split preconditioner, where $\mathbf{M} = \mathbf{M}_L\mathbf{M}_R$ and

$$\mathbf{M}_L^{-1}\mathbf{A}\mathbf{M}_R^{-1}\mathbf{u} = \mathbf{M}_L^{-1}\mathbf{b}, \text{ with } \mathbf{x} = \mathbf{M}_R^{-1}\mathbf{u}. \tag{2.22}$$

A suitable preconditioner improves the convergence, while being cheap in its construction, storage and application. It is, however, not trivial to construct robust preconditioners as multiple sources of ill-conditioning may exist. This work provides insight into preconditioning strategies in the context of the finite cell method and its extensions.

Comment on the notation

Now that the fundamental ingredients of the finite element method have been introduced, the notation used for certain terms will be simplified to foster readability. The finite element mesh will from here on be denoted by $\mathcal{T}$ instead of $\mathcal{T}_h$. From this point forward, we will only distinguish between global basis functions φ_i, that are defined over Ω, and element basis functions $\hat{\varphi}_i$, that are defined in the reference element coordinate system. Furthermore, an index set $\mathcal{I} = \{i \dots n_{\text{DOFs}}\}$ will be used, where each entry in $\mathcal{I}$ refers to a specific global basis function. This index set will later be beneficial when classifying the global degrees into different groups in the section on parallel computations.

2.2 Extensions of the finite element method

The quality of a finite element solution is strongly dependent on the spatial resolution of the mesh and the polynomial order of the elements it contains. Different error measures can be used to assess how well a finite element solution u_h approximates the true solution u. Common error measures include the L^2-error of the primal variable, $\|u - u_h\|_{L^2(\Omega)}$, and the error in the energy norm, $\|u - u_h\|_{E(\Omega)}$, which are defined as

$$\|v\|_{L^2(\Omega)} = \sqrt{\int_\Omega v^2 \, \mathrm{d}\Omega}, \tag{2.23}$$

$$\|v\|_{E(\Omega)} = \sqrt{a(v,v)}. \tag{2.24}$$

Several discretization strategies can be used to improve (extend) the approximation properties of a finite element mesh. These methods are commonly referred to as extensions of the finite element method and differ in *i)* the way in which the mesh is modified to reduce the discretization error of the approximation and *ii)* the rate at which the numerical solution approaches the true solution in a certain norm. The following section briefly introduces the three most prevalent finite element extensions, namely the h-extensions, p-extensions and hp-extensions. The interested reader is referred to [Yosibash, 2011] for a discussion on the convergence rates of these extensions for different problem classes.

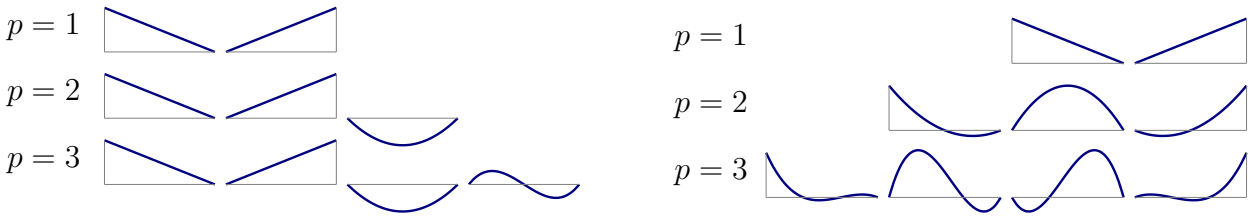
2.2.1 The h-version of the finite element method

The h-version of the finite element method (h-FEM) reduces the discretization error by increasing the spatial resolution of the mesh while leaving the polynomial orders of its elements unaltered. Elements with low polynomial orders, i.e. $p = \{1, 2\}$, are commonly employed in h-FEM. The spatial refinement, or h-refinement, can be applied either globally on all elements in the mesh or on a subset of elements in localized regions of the mesh. Methods that use global h-refinements are generally characterized by algebraic convergence rates in commonly used

error measures. Local h-refinements are usually performed in regions of the computational mesh associated with a large discretizational error. The choice of these crucial regions can be guided by error indicators, error estimators or a priori information such as specific geometrical features. An advantage of using local h-refinements as opposed to global h-refinements is that the number of unknowns does not drastically increase when capturing fine-scale solution characteristics. Numerical methods involving local h-refinements are, however, not trivial to implement since the presence of refined elements results in mesh irregularities which need to be resolved.

2.2.2 The p-version of the finite element method

In the p-version of the finite element method (p-FEM), a reduction of the discretizational error is achieved by elevating the polynomial order of the mesh elements while keeping the mesh's spatial resolution constant. A central aspect in p-FEM is the easy and efficient alteration of the mesh elements' polynomial orders. For this reason, it is common to employ elements with hierarchic shape functions. Since all hierarchic shape functions from order 1 up to order $p-1$ are contained in the hierarchic shape function set of order p, the elevation of an element's polynomial degree only involves the addition of new basis functions to previously defined functions. This is in contrast to non-hierarchic basis functions such as those based on Lagrange polynomials, where p-refinement results in the replacement of all previously defined basis functions with a set of completely new basis functions. The difference between hierarchic and non-hierarchic basis functions is illustrated for the one-dimensional case in Figure 2.2.



(a) Integrated Legendre basis (b) Lagrange basis

Figure 2.2: Comparison of the linear, quadratic and cubic one-dimensional Lagrange and integrated Legendre basis functions.

High-order finite elements based on the p-version of the finite element method proposed in [Babuška et al., 1981; Szabó and Babuška, 1991] play a key role in this thesis. The remaining part of this section summarizes central aspects of p-FEM and introduces concepts and terms important for understanding subsequent sections of this manuscript. The higher dimensional hierarchic element spaces introduced in [Szabó and Babuška, 1991] are constructed from a tensor product of one-dimensional integrated Legendre element spaces $\mathcal{S}^{p_i}$ as shown in (2.25), where the subscript i denotes the respective coordinate direction in the reference element. An element space is said to be *isotropic* if the same polynomial order is chosen in all coordinate directions and *anisotropic* in all other cases.

$$\begin{aligned} &\text{2D case :} \quad \mathcal{S}^{p_\xi,p_\eta} = \mathcal{S}^{p_\xi} \otimes \mathcal{S}^{p_\eta} \\ &\text{3D case :} \quad \mathcal{S}^{p_\xi,p_\eta,p_\zeta} = \mathcal{S}^{p_\xi} \otimes \mathcal{S}^{p_\eta} \otimes \mathcal{S}^{p_\zeta}. \end{aligned} \tag{2.25}$$

The one-dimensional element space $\mathcal{S}^p$ is spanned by the set of $p+1$ integrated Legendre polynomials defined as

$$\begin{aligned}\hat{\varphi}_1(\xi) &= \frac{1}{2}(1-\xi)\\ \hat{\varphi}_2(\xi) &= \frac{1}{2}(1+\xi)\\ \hat{\varphi}_{i+1}(\xi) &= P_i(\xi),\ i=2,\ldots,p,\end{aligned} \tag{2.26}$$

where $P_i(\xi)$ is the i^{th} integrated Legendre polynomial computed from the Legendre polynomials $L_{i-1}(\xi), L_i(\xi)$ and $L_{i-2}(\xi)$ via the formula,

$$P_i(\xi) = \sqrt{\frac{2i-1}{2}} \int_{-1}^{\xi} L_{i-1}(\xi)\ \mathrm{d}\xi = \frac{1}{\sqrt{4i-2}}(L_i(\xi) - L_{i-2}(\xi)),\ i=2,\ldots,p. \tag{2.27}$$

Bonnet's recursion formula can be used to compute the Legendre polynomials $L_i(\xi)$ for values of i larger than two and the start values $L_0(\xi)=0$ and $L_1(\xi)=1$.

$$L_i(\xi) = \frac{1}{i}[(2i-1)\xi L_{i-1}(\xi) - (i-1)L_{i-2}(\xi)],\ \forall i>2. \tag{2.28}$$

High-order integrated Legendre bases have several advantages. First, it was already mentioned that a p-refinement of a hierarchic basis can be readily realized by simply adding new basis functions. Secondly, using a hierarchic basis results in better conditioning of the discrete system than the standard Lagrange basis [Carey and Barragy, 1989]. An important property of the integrated Legendre basis is the direct association of each shape function to a specific topological component — node, edge, face, volume — of an element. This results in the classification of an element's shape functions, which are typically referred to as *modes*, into nodal, edge, face and volume modes. The number of modes associated with each component is determined by the order p and the type of function or ansatz space used to define the element. Different types of ansatz spaces can be constructed in multi-dimensional problems that differ in the completeness of the monomials used to span these spaces. Szabó and Babuška [1991] present two possible ansatz spaces for quadrilateral and hexahedral elements, the *tensor product space* $\mathcal{S}_{ps}$ and the *trunk space* $\mathcal{S}_{ts}$. Trunk spaces are similar to serendipity spaces and are spanned by reduced (truncated) sets of monomials resulting in fewer modes than the tensor product space [Szabó and Babuška, 1991].

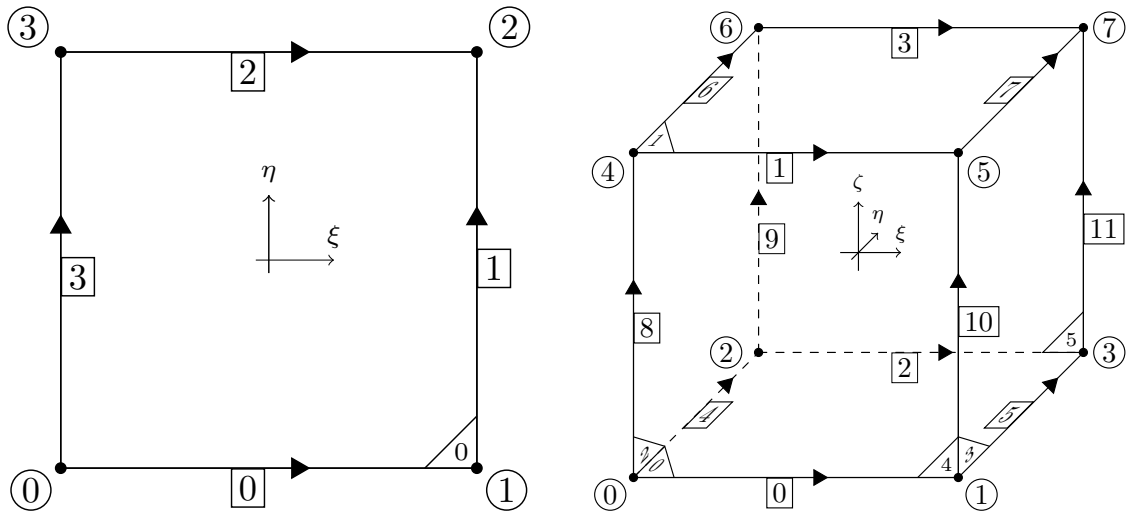
The different modes associated with the element topology are now presented for the tensor product and trunk spaces.

- **Nodal modes**: These functions are constructed by forming a tensor product of one-dimensional nodal functions. They are defined as

$$\text{2D case}: \quad \hat{\varphi}_i(\xi,\eta) = \frac{1}{4}(1+\xi_i\xi)(1+\eta_i\eta),\ i=1,\ldots,4 \tag{2.29a}$$

$$\text{3D case}: \hat{\varphi}_i(\xi,\eta,\zeta) = \frac{1}{8}(1+\xi_i\xi)(1+\eta_i\eta)(1+\zeta_i\zeta),\ i=1,\ldots,8, \tag{2.29b}$$

  with ξ_i, η_i and ζ_i denoting the local coordinates of the i^{th} node in the respective reference element.



(a) Quadrilateral element (b) Hexahedral element

Figure 2.3: Illustration of the standard quadrilateral and hexahedral p-FEM elements and the enumeration of their various topological components.

- **Edge modes**: Each edge is associated with $p_r - 1$ basis functions that assume their maximum values along this edge and are zero on all other edges and nodes. p_r is the polynomial order in the space direction of the edge with $r \in \{\xi, \eta, \zeta\}$ and $r \leq 2$. In order to construct an edge mode, a one-dimensional high-order mode is multiplied with either a single one-dimensional linear mode in a quadrilateral element or with two one-dimensional linear modes in a hexahedral element. The modes $\hat{\varphi}_i^{e_0}$ associated with edge 0 in Figures 2.3a and 2.3b read

$$\text{2D case}: \quad \hat{\varphi}_i^{e_0}(\xi,\eta) = \frac{1}{2}\hat{\varphi}_i(\xi)(1-\eta), \;\; i = 2,\ldots,p_\xi \tag{2.30a}$$

$$\text{3D case}: \hat{\varphi}_i^{e_0}(\xi,\eta,\zeta) = \frac{1}{4}\hat{\varphi}_i(\xi)(1-\eta)(1-\zeta), \;\; i = 2,\ldots,p_\xi. \tag{2.30b}$$

- **Face modes**: The functions associated with a given face assume their maximum values on that face and are zero on all other nodes, edges and faces. A face mode is constructed by the product of two one-dimensional high-order modes for a quadrilateral element or by multiplying two high-order 1D modes with one linear 1D mode for hexahedral elements. The modes $\hat{\varphi}_{i,j}^{f_0}$ associated with face 0 in figures 2.3a and 2.3b are

$$\text{2D case}: \quad \hat{\varphi}_{i,j}^{f_0}(\xi,\eta) = \hat{\varphi}_i(\xi)\phi_j(\eta), \tag{2.31a}$$

$$\text{3D case}: \hat{\varphi}_{i,j}^{f_0}(\xi,\eta,\zeta) = \frac{1}{2}\hat{\varphi}_i(\xi)\hat{\varphi}_j(\eta)(1-\zeta), \tag{2.31b}$$

with the following bounds for the indices i and j

tensor product space	trunk space
$i = 2, \ldots, p_\xi$	$i = 2, \ldots, p_\xi - 2$
$j = 2, \ldots, p_\eta$	$j = 2, \ldots, p_\eta - 2$
	$i + j = 4, \ldots, \max(p_\xi, p_\eta)$

Following the above definition of face modes, the number of modes associated with a single face with local coordinates r, s is $(p_r - 1)(p_s - 1)$ for tensor product space elements and $(\min(p_r, p_s) - 2)(\max(p_r, p_s) - 3)/2$ for trunk space elements.

- **Volume modes**: These functions are also called internal modes since they are only nonzero in the interior of an element. They are constructed by a tensor product of three one-dimensional high-order shape functions and are thus only defined for more than two spatial dimensions. The modes $\hat{\varphi}^{v_0}_{i,j,k}$ associated with the volume of a hexahedral element are defined as

$$\hat{\varphi}^{v_0}_{i,j,k} = \hat{\varphi}_i(\xi)\hat{\varphi}_j(\eta)\hat{\varphi}_k(\zeta), \tag{2.32}$$

with the following bounds for the indices i, j and k that differ for the tensor product and trunk spaces.

tensor product space	trunk space
$i = 2, \ldots, p_\xi$	$i = 2, \ldots, p_\xi - 4$
$j = 2, \ldots, p_\eta$	$j = 2, \ldots, p_\eta - 4$
$k = 2, \ldots, p_\zeta$	$k = 2, \ldots, p_\zeta - 4$
	$i + j = 6, \ldots, \max(p_\xi, p_\eta, p_\zeta)$

Each hexahedral element has $(p_\xi - 1)(p_\eta - 1)(p_\zeta - 1)$ volume modes for the tensor product space and $(\max(p_\xi, p_\eta, p_\zeta) - 5)(\text{med}(p_\xi, p_\eta, p_\zeta) - 4)(\min(p_\xi, p_\eta, p_\zeta) - 3)/6$ modes for the trunk space, where $\text{med}(\cdot)$ denotes the median of p_ξ, p_η and p_ζ.

2.2.3 The *hp*-version of the finite element method [1]

The *hp*-version of the finite element method (*hp*-FEM) combines an increase of the mesh spatial resolution, as performed in the *h*-version, with an adaptation of the element polynomial orders, as in the *p*-version, in crucial regions of the computational mesh. By combining the benefits of both *h*- and *p*-FEM, *hp*-FEM provides efficient discretizations associated with an improved approximation accuracy and comparatively low computational cost. It has been shown that this combination even allows exponential convergence rates to be recovered in singular problems [Babuška and Suri, 1990].

[1] The following section is based on [Jomo et al., 2017]. The main scientific research as well as the textual elaboration of the publication was performed by the author of this work.

Although *hp*-adaptive formulations possess superior approximation qualities, their implementational complexity has inhibited their widespread use in the field of computational mechanics. This complexity stems mainly from the algorithms and data structures needed to deal with mesh irregularities that arise during refinement and coarsening procedures. Classical *hp*-formulations commonly perform mesh refinement by replacing elements having a high discretization error with a set of smaller elements, *cf.* [Paszyński and Demkowicz, 2006; Šolín, 2004]. This process introduces mesh irregularities, usually referred to as hanging nodes, as the basis functions of the new elements lack a corresponding counterpart in their unrefined neighboring elements [Hughes, 2000]. Special algorithms are therefore needed to constrain these hanging nodes and restore inter-element continuity. Many *hp*-formulations, as a consequence, only allow one level of mesh irregularity between elements e.g. [Demkowicz, 2007; Paszyński and Demkowicz, 2006], while others implement sophisticated constraining algorithms that can deal with arbitrary levels of hanging nodes such as [Šolín et al., 2008, 2010].

Zander et al. [2015] propose a novel *hp*-adaptive approach based on refinement by superposition that circumvents the difficulties introduced by hanging nodes. By utilizing simple rule-sets to ensure compatibility and linear independence of the shape functions, intuitive refinement and coarsening procedures are developed. The need for complex data structures and algorithms to consolidate and constrain multiple levels of hanging nodes is alleviated as hanging nodes are treated naturally in the method. This *hp*-method plays a key role in this thesis and is explained in Section 3.2.

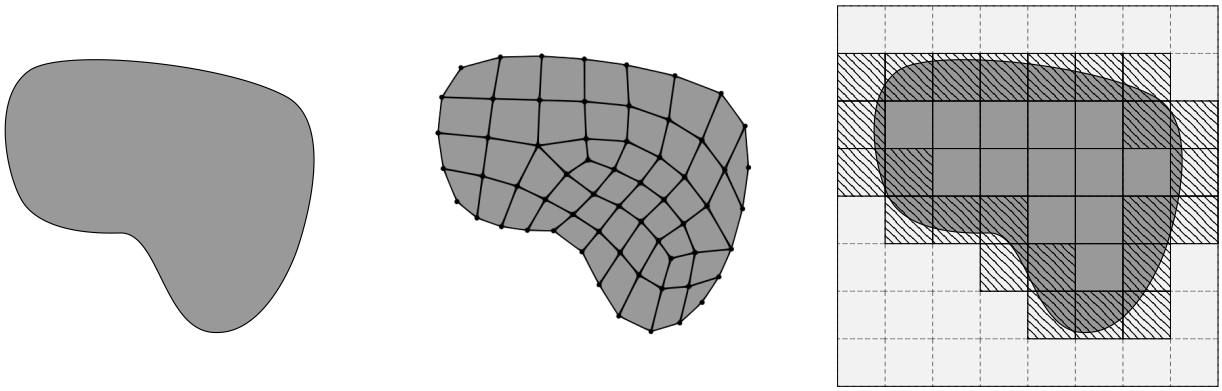
2.3 Immersed finite element methods

Immersed methods, also referred to as embedded or unfitted methods, are a class of finite element techniques that aim at providing a versatile and simple FE-analysis pipeline, by minimizing the effort needed for mesh generation. This is achieved by utilizing a non-boundary conforming discretization, i.e. a mesh that does not coincide with the boundary of the body being analyzed. This is in contrast to boundary-conforming or mesh-fitting finite elements, in which the boundary of the computational mesh captures, either perfectly or approximately, the boundary $\partial\Omega$. Generating boundary-conforming meshes can be a challenging task, especially when bodies with complex geometries are considered. The effort associated with this task is further increased in the case of high-order finite elements, where different blending operations may be required to achieve an acceptable representation of the complex boundary. Such operations can be time-consuming as they may require manual intervention and/or a chain of meshing operations, potentially involving different software packages, to obtain an analysis-suitable discretization.

2.3.1 The core idea behind immersed finite elements

Immersed methods avoid computationally demanding mesh generation steps by using a background discretization that is independent of the shape of the domain Ω. The name "immersed" originates from the fact that Ω is enclosed by (or immersed in) the background mesh. This immersion results in three kinds of elements: *i)* firstly elements completely within the geometry of the original body, termed inside elements, denoted in this manuscript by K^{in}, *ii)* elements completely outside Ω termed as outside or inactive elements, K^{out}, and *iii)* so-called cut or

trimmed elements, K^{cut}, which as the name suggests are intersected (cut) by the boundary $\partial\Omega$. In general only inside and cut elements, collectively referred to as active elements K^{act}, are considered during a simulation in immersed approaches.



(a) Original domain Ω. (b) Boundary-conforming mesh. (c) Immersed mesh.

Figure 2.4: Illustration of the general idea behind immersed finite element methods. A body Ω with an arbitrary shape is placed in a surrounding background mesh resulting in active, cut and inactive elements. This is in contrast to boundary-conforming finite elements, whose element topology matches the domain boundary $\partial\Omega$.

2.3.2 Cut cells and their implications

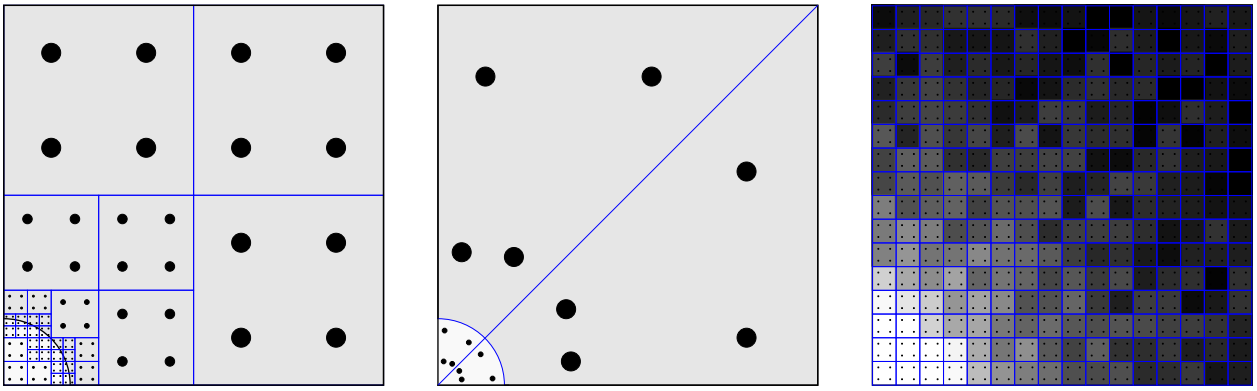
Although mesh generation is trivial in the context of immersed methods, the presence of cut elements has several implications on the subsequent stages of the finite element pipeline. The three major aspects affected by cut elements include:

1. The numerical quadrature on cut elements.
2. The application of Dirichlet boundary conditions.
3. The conditioning of the system and the stability of the immersed method.

Proper handling of these three aspects is crucial in order to exploit the full benefits of immersed methods. Favorable properties of boundary-conforming FE-methods such as optimal convergence rates or the scaling of the condition number with respect to the mesh size are not naturally present in immersed methods. These properties are only recovered when certain aspects of the cut cells are adequately addressed. The following section briefly reviews some of the strategies developed to deal with cut cells in the areas of numerical quadrature, boundary condition imposition and conditioning. These three areas have been the subject of several publications and remain vibrant research topics to date.

2.3.2.1 Dedicated numerical quadrature rules for cut cells

Standard quadrature rules are not accurate when directly applied on cut cells due to the discontinuous integrands that result from the presence of the immersed boundary. For this reason, more elaborate quadrature schemes are required to adequately approximate/capture



(a) Recursive quadtree scheme. (b) Blended partitioning. (c) Voxel integration.

Figure 2.5: Illustration of three commonly used cut-element integration techniques: Figure 2.5a shows a recursive quadtree scheme, Figure 2.5b shows a conformal decomposition scheme by means of blended partitioning [Kudela et al., 2015] and Figure 2.5c depicts a voxel-based integration scheme.

integrals on cut cells. Numerous cut-cell integration schemes exist and [Düster et al., 2017] gives an overview of the different strategies that can be applied. In this work, focus is placed on schemes based on the partitioning of a cut cell into separate integration subdomains, followed by the application of the quadrature rule separately on each subdomain. This form of integration is referred to as *composed integration* [Düster et al., 2008; Parvizian et al., 2007] since an integral over a cut cell is approximated by a sum of integrals over the subdomains. Equation (2.33) illustrates the concept of composed integration for a single one-dimensional finite element with $x \in [a, b]$ partitioned into n_{sub} non-overlapping integration domains. Each subdomain has a local coordinate system $r_i \in [-1, 1]$ and a Jacobian matrix $\mathbf{J}_{r_i}$ that transforms entities from the local coordinate system of the i^{th} subdomain to the coordinate system of the reference element. The number of quadrature points in each subdomain is denoted by n_{GP} whereas $r_{i,j}$ denotes their position and w_j their respective weights.

$$\int_a^b f(x)\,\mathrm{d}x = \int_{-1}^{1} f(\xi)|\mathbf{J}_K|\,\mathrm{d}\xi = \sum_{i=1}^{n_{\text{sub}}} \int_{-1}^{1} f(r_i)\,|\mathbf{J}_{r_i}||\mathbf{J}_K|\,\mathrm{d}r_i = \sum_{i=1}^{n_{\text{sub}}} \sum_{j=1}^{n_{\text{GP}}} f(r_{i,j})\,w_j|\mathbf{J}_{r_i}||\mathbf{J}_K|. \quad (2.33)$$

Several composed integration techniques have been developed, which mainly differ in the generation and topology of the integration subdomains and in the quadrature rules applied on each subdomain. The union of all integration subdomains is called the integration mesh. This integration mesh does not introduce additional unknowns and needs not adhere to strict requirements, such as continuity over element edges and faces, and can thus contain hanging nodes, cells of different types, or even cells with high aspect ratios [Hubrich et al., 2017]. Gauss quadrature is typically applied on the integration subdomains. Some approaches combine composed integration with moment-fitting, where the latter is used to generate the position and weight of the integration points in the different subdomains [Hubrich et al., 2017; Joulaian et al., 2016].

Recursive space-tree decomposition

The most prevalent approach for the generation of integration subdomains is the use of recursive space-trees, i.e. quadtrees in two dimensions and octrees in three dimensions. Space-trees recursively refine the integration mesh towards the immersed boundary until a specified depth, see Figure 2.5a. They can deal with arbitrary cut configurations and are widely used in conjunction with standard Gauss quadrature due to their robustness [Düster et al., 2008]. This approach is not only robust but can be easily automatized and allows the control of the integration error [Hubrich et al., 2017]. Integration schemes based on space-tree decomposition of cut cells usually have a higher computational cost for the same level of integration error than other schemes. This is due to the high number of integration points generated. A recent publication shows that integration points can be more effectively distributed over the integration subdomains of a space-tree using an adaptive approach based on error-estimates [Divi et al., 2020]. This strategy can help reduce the computational effort associated with space-tree type integration procedures.

Conformal decomposition of cut cells

Standard space-tree decomposition yields a 0^{th}-order approximation of the immersed boundary in cut cells. Different strategies can be employed to achieve a better approximation of the immersed boundary. First-order boundary approximations can be achieved by a tessellation of the cut cells, a concept that originated from XFEM [Sukumar et al., 2000]. This tessellation can either be applied directly on the cut cell, or combined with a space-tree approach and applied on the highest level of the space-tree as in [Verhoosel et al., 2015]. Other integration approaches decompose cut cells into high-order subdomains, which provide a high-order approximation of the immersed boundary such as [Cheng and Fries, 2010; Kudela et al., 2015, 2016], see Figure 2.5b. Although integration schemes based on conformal decompositions yield accurate results with a smaller number of integration points compared to the standard space-tree approach, they are less robust than the space-trees and require knowledge of the position and shape on the immersed boundary to calculate intersection points.

Voxel-based integration schemes

The simplest way of performing numerical quadrature on a cut element is by partitioning it using a uniform grid into integration subdomains and performing standard Gauss quadrature on each subdomain as illustrated in Figure 2.5c. Consider the integration of the element stiffness matrix $\mathbf{K}_e$ in a linear elastic problem, in which the element domain Ω_e is divided into $v_x \times v_y \times v_z$ voxels. The voxel-wise computation of $\mathbf{K}_e$ can be written as

$$\begin{aligned}
\mathbf{K}_e &= \int_{\Omega_e} \mathbf{B}^T \alpha \mathbf{C} \mathbf{B} \, \mathrm{d}\Omega_e = \sum_{l=0}^{v_x} \sum_{n=0}^{v_y} \sum_{m=0}^{v_z} \int_{\Omega_v} \mathbf{B}^T \alpha \mathbf{C} \mathbf{B} \, \mathrm{d}\Omega_v \\
\mathbf{K}_e &= \int_{\Omega_e} \mathbf{B}^T \alpha \mathbf{C} \mathbf{B} \, \mathrm{d}\Omega_e = \sum_{l=0}^{v_x} \sum_{n=0}^{v_y} \sum_{m=0}^{v_z} \left(\sum_{i=0}^{n_{\mathrm{GP}_i}} \sum_{j=0}^{n_{\mathrm{GP}_j}} \sum_{k=0}^{n_{\mathrm{GP}_k}} \mathbf{B}^T \alpha \mathbf{C} \mathbf{B} \; w_i w_j w_k |\mathbf{J}_v| |\mathbf{J}_{\mathbf{K}_e}| \right)
\end{aligned} \tag{2.34}$$

where Ω_v and $|\mathbf{J}_v|$ denote the domain and Jacobian matrix of a single voxel, respectively. The term $\mathbf{B}$ represents the linear strain operator and $\mathbf{C}$ denotes the material matrix. This

approach is commonly applied in image-based immersed analyses involving CT scans e.g. [Korshunova et al., 2020; Ruess et al., 2012], where each integration subdomain encloses a predefined number of voxels. The structure and simplicity of this approach allows the use of different kinds of optimization techniques that accelerate the computation of element matrices. In [Yang et al., 2012], the matrix $\mathbf{K}_e$ is pre-computed for linear elastic problems leading to significant reductions in the computational time. Schillinger and Ruess [2014] use shared-memory parallelism to accelerate the computation of the $\mathbf{K}_e$ following 2.34 and also investigate the benefits of pre-computation. Korshunova et al. [2020] combine the pre-computation of $\mathbf{K}_e$ with the parallel algorithms developed in this thesis for the computataion of large finite cell systems based on CT scans. An example of one of these computations is shown in Section 6.1.1.

2.3.2.2 Weak imposition of Dirichlet boundary conditions

Homogeneous Neumann boundary conditions are naturally satisfied in immersed methods while their inhomogeneous counterparts are applied in the same manner as in boundary conforming methods by integrating over the Neumann surfaces. Dirichlet boundary conditions, however, need special treatment in immersed methods since the domain boundary $\partial\Omega$ does not coincide with mesh entities, such that the traditional strong imposition of boundary conditions is not applicable. It is common to apply Dirichlet boundary conditions weakly using variational techniques that are based on a modification of the weak form in immersed methods. The following section gives a brief description of three of the most prevalent methods.

Lagrange multiplier method

The Lagrange multiplier method reformulates the application of boundary conditions as a constrained minimization problem. In this method, the Dirichlet boundary conditions play the role of the constraints that are enforced through an additional field of unknowns attributed to the Lagrange multipliers. The original weak formulation $a(u,v) = f(v)$ is thus transformed into a two-field problem that is made up of the unknown variables u, the Lagrange multiplier field λ_{LM} and their corresponding test functions v and μ as shown in (2.35). g_D denotes the values prescribed along the Dirichlet boundaries.

$$a(u,v)_\Omega - (\lambda_{LM}, v)_{\partial\Omega} = f(v) \tag{2.35a}$$

$$(u,\mu)_{\partial\Omega} = (g_D,\mu)_{\partial\Omega}. \tag{2.35b}$$

The major advantage of the Lagrange multiplier method is its generality, i.e. the approach can be readily applied to different types of PDEs [Fernández-Méndez and Huerta, 2004]. One of the method's main drawbacks is the introduction of additional unknowns. Moreover, the discretized form in the Lagrange multiplier method leads to a saddle point problem. The finite element spaces associated with the trial and test functions in this approach therefore have to be specially chosen to guarantee inf-sup stability. Examples of immersed grids that use this method include [Burman and Hansbo, 2010; Gerstenberger and Wall, 2010; Glowinski and Kuznetsov, 2007].

The penalty method

This is the simplest method for the weak enforcement of boundary conditions [Babuška, 1973]. It involves the introduction of a penalty term $\beta(u - g_D, v)$ in the weak form, that controls the enforcement of the Dirichlet boundary conditions through the value of the penalty parameter β. The penalty method can be interpreted as a regularization of the Lagrange multiplier method and is widely used because it does not introduce additional unknowns, is easy to implement, has a low computational cost, maintains the symmetry and positive definiteness of the system when discretized and does not require modification for different problem types. The modified weak form in the penalty method can be written as

$$a(u, v) + \beta(u, v)_{\partial\Omega} = f(v) + \beta(g_D, v)_{\partial\Omega}. \tag{2.36}$$

Sufficiently large penalty parameters are required to guarantee proper enforcement of the boundary conditions. The penalty method is asymptotically consistent since the exact solution is only recovered when β tends to infinity. High penalty values can, however, have a negative effect on the conditioning of the system. Moreover, a proper scaling of the penalty parameter with respect to the mesh size h is needed in order to maintain optimal convergence orders. In [Babuška, 1973], the penalty parameter is chosen proportional to $h^{-\frac{2p+1}{3}}$ resulting in a rate of convergence in the energy norm of order $h^{\frac{2p+1}{3}}$. The penalty method is widely used in the finite cell community e.g. [Schillinger and Ruess, 2014] and is applied in this thesis.

Nitsche's method

Nitsche [1971] proposed a variationally consistent method for the weak imposition of Dirichlet boundary conditions that does not add any additional unknowns. There are different ways of interpreting Nitsche's method and it is often viewed as a penalty method that contains additional terms that ensure variational consistency for finite values of β [Fernández-Méndez and Huerta, 2004] but can be also viewed as a stabilized method that is related to stabilization techniques in Lagrange multiplier methods [Stenberg, 1995]. Two variants of Nitsche's method are generally used in immersed finite element methods, a symmetric version whose weak form for the Poisson equation is given by

$$\begin{aligned} a(u, v) - (n \cdot \Delta u, v)_{\partial\Omega} - (u, n \cdot \Delta v)_{\partial\Omega} + \beta(u, v)_{\partial\Omega} \\ = f(v) - (g_D, n \cdot \Delta v)_{\partial\Omega} + \beta(g_D, v)_{\partial\Omega}, \end{aligned} \tag{2.37}$$

and a non-symmetric version whose corresponding weak form reads

$$a(u, v) - (n \cdot \Delta u, v)_{\partial\Omega} + (u, n \cdot \Delta v)_{\partial\Omega} = f(v) - (g_D, n \cdot \Delta v)_{\partial\Omega}. \tag{2.38}$$

The non-symmetric version of Nitsche's method [Burman, 2012] does not require stabilization, i.e. penalty terms that are added to guarantee coercivity of the weak form as opposed to its symmetric counterpart. The symmetric version is, however, more prevalent in literature, one of the reasons being that it maintains the symmetry of positive-definite systems and does not require solvers for non-symmetric matrices. The stabilization parameter in this version is chosen in proportion to the mesh size with $\beta \propto h^{-1}$. The value of β can be set globally by intuition or by solving a global eigenvalue problem [Griebel and Schweitzer, 2003] or set locally for every cell with a Dirichlet boundary by solving local eigenvalue problems as proposed in

[Embar et al., 2010]. A major advantage of Nitsche's method over the penalty method is that it does not require large penalty values in order to ensure correct enforcement of the boundary conditions. Applying Nitsche's method to general problems is, however, challenging since the consistency terms need to be consistently treated e.g. the consistency terms need to be correctly linearized in nonlinear problems.

2.3.2.3 Conditioning of the system[2]

It is well known that small cut cells in immersed methods deteriorate the conditioning of the arising systems, a phenomenon commonly referred to as the *small cut cell problem*. De Prenter et al. conduct a systematic study of the conditioning of immersed FE methods in de Prenter et al. [2017] and identify the root cause of ill-conditioning in these methods as a combination of the presence of small basis functions and the occurrence of basis functions that become almost linear dependent when elements are cut. They derive a scaling relation between the condition number $\kappa(\mathbf{A})$ and the smallest volume fraction η defined as the smallest relative intersection of the physical domain Ω_{phys} and a mesh element K with

$$\eta = \min_{K \in \mathcal{T}_h} \frac{|K \cap \Omega_{\text{phys}}|}{|K|}. \tag{2.39}$$

This relation is derived under the assumption that the cut elements are shape regular, i.e. elements that are non-degenerate with $\det(\mathbf{J}_K) > 0$. This relation is given as

$$\kappa\left(\mathbf{A}\right) \propto \eta^{-(2p+1-2/d)}, \tag{2.40}$$

with d the spatial dimension of the problem and p the polynomial order of the discretization. The interested reader is referred to [de Prenter et al., 2017] for a derivation of this relation.

Different strategies can be adopted to reduce, avoid or even eliminate the conditioning problems related to cut elements. The following section provides a brief overview of some of the commonly used methods for treating conditioning problems in immersed methods. Note that the mentioned strategies can also be combined to improve the convergence of iterative solvers when solving immersed systems.

Preconditioning

Tailored preconditioners can be used to remedy the conditioning problems resulting from immersed boundaries. This strategy has been employed for different immersed methods and for similar problems in the extended finite element method (XFEM) [Belytschko and Black, 1999; Moës et al., 1999]. In XFEM, these preconditioners generally split the matrix in *(i)* non-problematic DOFs, which can be treated by standard preconditioning techniques and *(ii)* problematic DOFs, that are treated separately by local Cholesky decompositions [Béchet et al., 2005], a tailored FETI-type method [Menk and Bordas, 2011], a Schur complement based algebraic multigrid preconditioner [Hiriyur et al., 2012] or a Schwarz-type domain decomposition preconditioner [Berger-Vergiat et al., 2012; Waisman and Berger-Vergiat, 2013]. The last two references are conceptually similar to the approach applied in this work, but

[2]The following section is based on [Jomo et al., 2019]. The main scientific research as well as the textual elaboration of the publication was performed by the author of this work.

differ in the choice of subdomains as the problematic DOFs in the finite cell method, i.e. all DOFs on the boundary, are generally large connected sets. For immersed finite element methods it is proposed in [Lehrenfeld and Reusken, 2017] to diagonally scale the non-problematic DOFs and treat the problematic DOFs by an algebraic multigrid procedure. This strategy is demonstrated to be effective for linear basis functions but has restrictions on the manner in which elements can be cut. In [Badia and Verdugo, 2017] the scaling in a Balancing Domain Decomposition by Constraints preconditioner (BDDC) is customized for cut basis functions, which is shown to result in an effective method for linear bases. A preconditioner that combines a diagonal scaling with a local orthonormalization of the problematic DOFs is developed in [de Prenter et al., 2017]. In de Prenter et al. [2019a] it is shown that this orthonormalization is very similar to additive Schwarz preconditioning, e.g. [Brenner and Scott, 2008], of the cut elements. The preconditioner developed in [de Prenter et al., 2019a] is demonstrated to be effective for immersed finite element methods with high-order basis functions. A modification of this preconditioner is applied to the multi-level *hp*-basis in [Jomo et al., 2019] and applied in a distributed parallel setting. de Prenter et al. [2019b] present a geometric multigrid preconditioner that robustly deals with cut cells in *hp*-refined problems and leads to convergence rates of iterative solvers that are independent of the mesh size.

Fictitious domain stiffness

In immersed boundary methods a differentiation is made between the physical domain Ω_{phys}, i.e. the domain defined by the original body Ω, and the non-physical or fictitious domain Ω_{fict}, made up of all points that do not belong to Ω_{phys}. A simple but effective way of stabilizing cut elements is to assume a soft material in the fictitious domain. This introduces an artificial stiffness in the system by evaluating integrals in cut cells for points lying within the physical and the fictitious domain. The amount of added stiffness is controlled by a scalar parameter α, which is usually chosen as a small constant $0 < \alpha \ll 1$. The inclusion of minimal stiffness improves the conditioning of the system and can be interpreted as a volumetric stabilization. [Dauge et al., 2015] provides an extensive analysis of this method. Fictitious domain stiffness is widely used in the finite cell method [Düster et al., 2008; Parvizian et al., 2007] and is commonly used together with different preconditioning techniques [Jomo et al., 2019].

Ghost penalty

Ghost penalty [Burman, 2010; Burman et al., 2014a; Burman and Hansbo, 2012] remedies the conditioning problems associated with cut cells by the addition of penalty terms at the interface of the cut and fully supported elements. This stabilization technique introduces a weak coupling between degrees of freedom in cut cells and their fully supported adjacent elements hereby extending the coercivity of the physical domain to the cut elements. Although the ghost penalty method effectively ameliorates condition problems associated to cut cells, it leads to an increased sparsity pattern that increases the effort needed to solve the system. Ghost penalty can be efficiently applied in the context of isogeometric analysis since its implementation is simplified by the high-order regularity of the B-spline basis. Ghost penalty has been successfully applied in different scenarios such as in flow problems [Burman et al., 2014a; Massing et al., 2018].

Basis function manipulation

An alternative way of dealing with the ill-conditioning in immersed methods is by eliminating the problematic basis functions only supported on small cut cells. This can be done in different ways. One approach is to simply exclude problematic basis functions with small support inside the domain of computation from the linear system. This strategy remedies ill-conditioning and also bounds the stabilization parameter in Nitsche's method. It is applied in [Elfverson et al., 2018] without affecting the accuracy of the solution and also in [Verhoosel et al., 2015]. The effect of cut cells can also be eliminated by strongly coupling problematic basis functions to neighbor elements as done in [Badia et al., 2018b; Höllig et al., 2005; Höllig et al., 2001; Marussig et al., 2018; Verdugo et al., 2019].

2.3.3 A brief overview of different immersed methods

The ability of immersed finite element methods to deal with a wide range of geometric models makes them a suitable choice for performing numerical analyses on complex geometries. Several immersed methods have been developed in the past decades. These techniques differ mainly in the means by which Dirichlet boundary conditions are applied, the type of cut cell integration used, the way in which the conditioning of the system is improved and the type of basis functions they employ. The most prominent immersed methods include

- The finite cell method, **FCM**, is an immersed method that combines a fictitious domain approach with high-order finite elements and was introduced by Parvizian et al. [2007] in the two-dimensional setting and extended to three dimensions in [Düster et al., 2008]. The work in this thesis is centered on the finite cell method and a detailed description of the method is given in Section 3.1.
- The *Cut Finite Element Method*, or **CutFEM** in short, is an immersed FE method attributed to Burman and Hansbo [2012]. Although different flavors of this method exist, the most common ingredients in CutFEM-type methods include the application of essential boundary conditions using Nitsche's method and the use of the ghost penalty to treat the ill-conditioning of cut cells. This combination allows the use of a global stabilization parameter that is independent of the cut elements. CutFEM has numerous applications fields such as various flow problems [Burman et al., 2014b; Burman and Hansbo, 2012; Massing et al., 2018], fluid-structure interaction [Ager et al., 2019; Schott et al., 2019] and topology optimization [Burman et al., 2019].
- The aggregated unfitted method, **AgFEM**, is a novel immersed technique that is based on the removal of problematic basis functions through the use of specially designed constraints [Badia et al., 2018b]. These constraints ameliorate the ill-conditioning due to cut cells and allow the application of standard solvers and preconditioners e.g. algebraic multigrid preconditioners as demonstrated in [Verdugo et al., 2019].
- The Cartesian grid Finite element method, abbreviated as **cgFEM**, is an embedded method based on Cartesian background grids. By taking advantage of the structure of the embedding mesh, it is shown that the method can be implemented efficiently and results in low computation times. The method has been applied in linear elastic simulations [Nadal Soriano et al., 2013], image-based analyses [Giovannelli et al., 2014] and immersed contact mechanics [Navarro-Jiménez et al., 2018].

- The shifted boundary method [Main and Scovazzi, 2018a], is a recently introduced embedded domain method that is based on an approximate/surrogate boundary algorithm. The core idea behind this method entails the shifting of the location at which boundary conditions are applied from the immersed boundary to surfaces that correspond to element boundaries. The original boundary conditions are modified accordingly to allow the weak enforcement on the new boundary-conforming surfaces and to preserve optimal convergence rates of the numerical method. This method eliminates the conditioning problems related to cut cells and has been successfully applied to numerical tests for the Poisson and Stokes equations [Main and Scovazzi, 2018a] as well as the advection-diffusion equation and the Navier-Stokes equation [Main and Scovazzi, 2018b].

Although different immersed methods exists, these approaches have more similarities than differences. In fact, most if not all of the algorithms presented in Section 2.3.2 can be applied to all immersed methods without requiring significant modifications. The most commonly used method for the weak imposition of Dirichlet boundary conditions is Nitsche's method, while the most prevalent method for dealing with small cut cells is either preconditioning or some form of basis function manipulation.

Chapter 3

Formulation of the multi-level hp-finite cell method

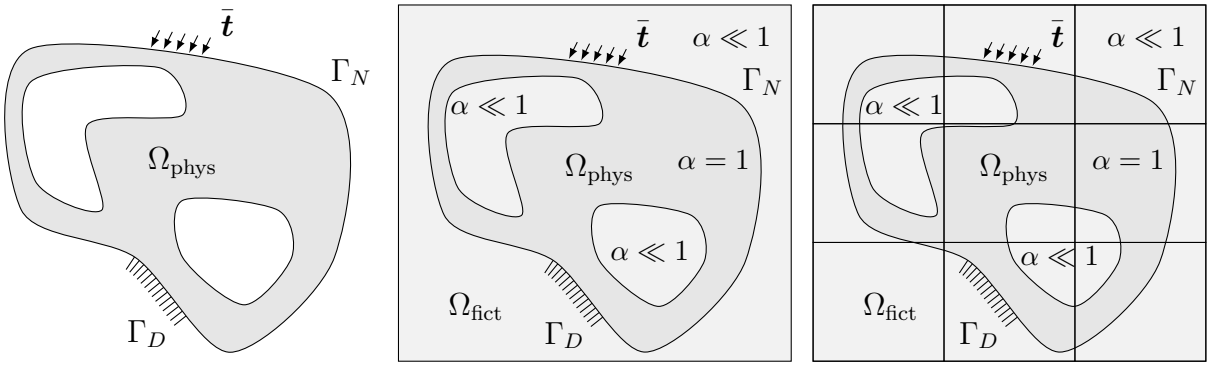
In this chapter, the discretization techniques that are central to this work, namely the finite cell method and multi-level hp-refinement, are presented. These numerical methods have been jointly used in a wide range of engineering applications involving bodies with complex geometries. These application areas include cohesive fracturing modeling [Zander et al., 2016b], stress analysis of structures such as bone-implant systems [Elhaddad et al., 2018] and mechanical engineering artifacts manufactured by die casting [Jomo et al., 2019], the simulation of metal additive manufacturing processes [Kollmannsberger et al., 2019, 2018; Özcan et al., 2019] and the modeling of fracture growth in brittle structures using a phase-field approach [Hug et al., 2020; Nagaraja et al., 2019].

In this thesis, the finite cell method is used in conjunction with multi-level hp-refinement for the efficient numerical analysis of geometrically and topologically complex domains. Alternative approaches presented in literature combine FCM with hierarchical B-splines thus merging the benefits of immersed methods with those of isogeometric analysis. This methodology has been successfully applied in the context of (non-linear) elasticity [Schillinger et al., 2012] and in image-based analysis of trabecular bone [Verhoosel et al., 2015].

3.1 Fundamentals of the finite cell method

The finite cell method (FCM) is a well-known immersed finite element method first introduced in [Parvizian et al., 2007]. It combines the favorable approximation properties of high-order finite elements with a fictitious domain concept yielding a flexible and accurate discretization scheme suitable for FE-analysis of bodies with a complex geometry. Various types of high-order shape functions can be utilized in constructing a FCM mesh, such as the integrated Legendre polynomials commonly used in p-FEM [Düster et al., 2008; Parvizian et al., 2007], B-splines or NURBS [Kamensky et al., 2015; Schillinger and Rank, 2011a] and Lagrange polynomials based on Gauss-Lobatto nodal distributions [Joulaian et al., 2014; Schillinger and Rank, 2011b]. Note that any admissible set of high-order basis functions used in boundary-conforming finite element frameworks can be incorporated into the finite cell setting.

In this thesis, the commonly used integrated Legendre basis functions presented in Section 2.2.2 are utilized. This choice is motivated by the functions' hierarchical nature that simplifies



(a) Physical domain Ω_{phys} (b) Fictitious extension (c) Structured discretization

Figure 3.1: Schematic representation of the finite cell method adopted from [Bog et al., 2017].

the p-adaptation of mesh elements. All theoretical considerations and numerical examples in this thesis are, therefore, based on this class of basis functions. The developed concepts can, however, be extended to different basis function classes, albeit with certain modifications.

The finite cell method provides a flexible and accurate discretizational framework. An original domain of complex shape, usually termed the physical domain Ω_{phys}, is extended by or immersed in a fictitious domain, Ω_{fict}, such that their union yields a computational domain $\Omega_{\cup}$ of simple shape that can be trivially discretized using regular elements. As a consequence, the effort associated with mesh generation in FCM is negligible.

Another implication of this fictitious extension is that the weak form in FCM is no longer posed solely on $\Omega = \Omega_{\text{phys}}$ but rather on the extended computational domain $\Omega_{\cup} = \Omega_{\text{phys}} \cup \Omega_{\text{fict}}$. In order to differentiate between points lying in Ω_{phys} and Ω_{fict}, an indicator function α is introduced which associates each material point $\boldsymbol{x}$ with its respective domain such that

$$\alpha(\boldsymbol{x}) = \begin{cases} 1 & \forall\, \boldsymbol{x} \in\ \Omega_{\text{phys}} \\ \ll 1 & \forall\, \boldsymbol{x} \in\ \Omega_{\text{fict}} \end{cases}. \tag{3.1}$$

The indicator function $\alpha(\boldsymbol{x})$ is applied to the original weak form to penalize the contributions of the fictitious domain. The strength of this penalization is controlled by the value of α and is usually taken to be a small constant $0 < \alpha \ll 1$ instead of $\alpha = 0$ to ensure numerical stability. The modified weak form in the context of the finite cell method can be written as

$$a(u,v)_{\Omega} + \alpha \cdot a(u,v)_{\Omega_{\text{fict}}} + bc(u,v)_{\partial\Omega_D} = (f,v) + bc(g_D,v)_{\partial\Omega_D} + (g_N,v)_{\partial\Omega_N} \tag{3.2}$$

where $bc(u,v)_{\partial\Omega_D}$ and $bc(g_D,v)_{\partial\Omega_D}$ denote the contributions of the weak boundary conditions to the system matrices and force vector respectively, see Section 2.3.2.2. It should be noted that the indicator function introduces minimal artificial stiffness in the fictitious domain in order to improve the conditioning of the system. This additional stiffness is associated with a modeling error that is proportional to $\sqrt{\alpha}$. In [Dauge et al., 2015] it is shown that FCM retains the optimal convergence properties of high-order finite elements and is an asymptotically consistent method, since the original weak form is recovered when α tends to zero in the fictitious domain.

3.2 Multi-level *hp*-refinement

Section 2.2.3 presented the main goal of *hp*-methods, which is combining *h*-refinement and *p*-refinement to yield high-quality finite element discretizations capable of capturing fine-scale solution characteristics in localized regions of the computational mesh. Classical *hp*-methods perform mesh refinement based on the replacement of elements and introduce mesh irregularities called hanging nodes in the process, due to the incompatibility of the basis functions of the newly introduced small elements with existing basis functions of adjacent unrefined elements. Although hanging nodes can be treated in a systematic manner through specially designed constraints, implementing these algorithms is quite challenging especially for multiple refinement levels.

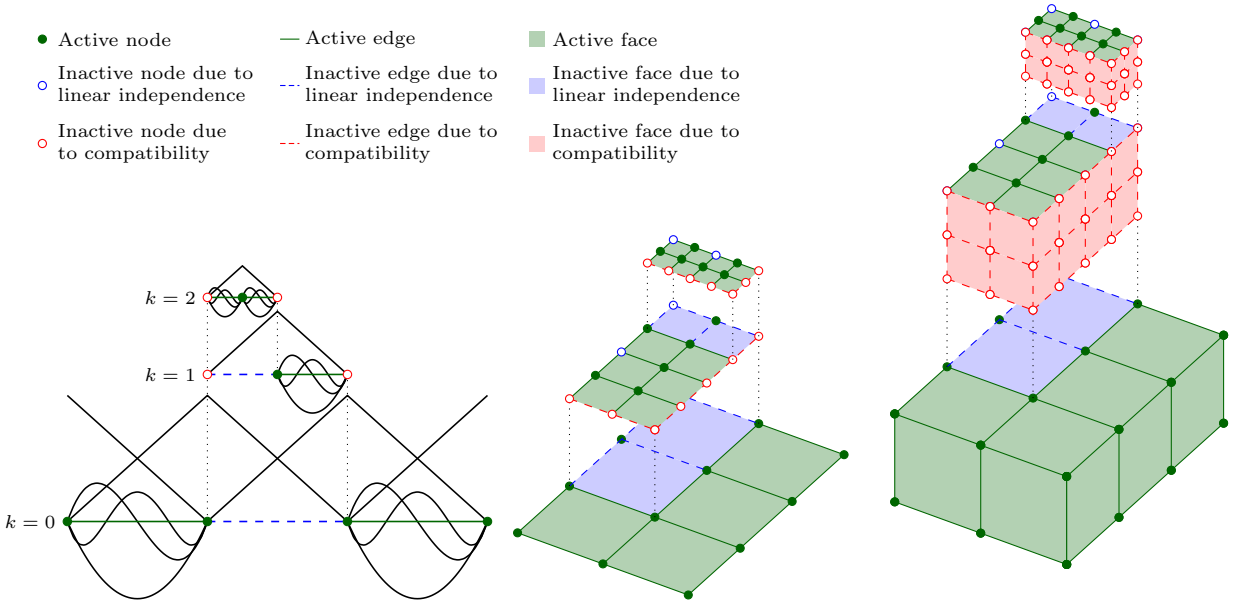
Multi-level *hp*-refinement, introduced in [Zander et al., 2015], is a novel *hp*-scheme that performs spatial refinement based on superposition rather than replacement. The scheme was developed in order to circumvent the challenges associated with constraining arbitrary levels of hanging nodes, while at the same time preserving the desirable characteristics of classical *hp*-formulations [Zander, 2017]. Furthermore, the scheme relies on intuitive procedures for refinement and coarsening, that not only simplify the implementation but also make the scheme suitable for dealing with complex dynamic mesh refinement scenarios in multiple dimensions. In this work, spatial refinement in the multi-level *hp*-method is driven by a priori information such as geometrical features but error estimators could also be employed as in [D'Angella et al., 2016].

3.2.1 Construction of the basis

The idea of *refinement by superposition* was first introduced by Mote in [Mote, 1971] and has since then be adopted in several refinement schemes such as [Belytschko et al., 1990; Moore and Flaherty, 1992; Rank, 1992; Schillinger and Rank, 2011a]. In multi-level *hp*-refinement, spatial refinement is carried out by superposing elements in a coarse base mesh with multiple levels of hierarchical overlay meshes. Arbitrary levels of hanging nodes are naturally treated through the use of homogeneous Dirichlet boundary conditions at the boundary of the overlay meshes. This process ensures the global C^0-continuity of the *hp*-mesh and with it the compatibility of the discretization. Linear independence of the basis functions can also be readily enforced through the deactivation of specific basis functions following a simple ruleset, that can be easily extended to multiple dimensions [Zander, 2017; Zander et al., 2016a, 2015]. This deactivation can be done in a straightforward manner by taking advantage of the direct relation between the basis functions and the element topology. Further details on the construction and implementation of the multi-level *hp*-basis can be found in [Zander, 2017]. Figure 3.2 illustrates the construction of the multi-level *hp*-mesh for different space dimensions.

3.2.2 Nomenclature and properties of a multi-level *hp*-mesh

A multi-level *hp*-mesh comprises elements with different refinement levels as shown in Figure 3.2. The letter k denotes the refinement level/depth of an element. Elements on the lowest refinement level are referred to as *base elements* and have a refinement depth $k = 0$. When performing *h*-refinement, a base element is superposed by subelements formed by a uniform bisection of the base element. This process is repeated recursively until the desired refinement



(a) One-dimensional case (b) Two-dimensional case (c) Three-dimensional case

Figure 3.2: Illustration of the multi-level *hp*-refinement scheme with two refinement levels, $k = 2$, in different spatial dimensions. The deactivation of specific topological components following a simple rule-set ensures compatibility and linear independence of the basis functions, [Zander et al., 2016a].

depth, resulting in a refinement tree for every base element. Generation of subelements by uniform bisections guarantees that each sub-element is associated with only one parent element. Each refined (parent) element has either two subelements in the one-dimensional case, four subelements in the case of quadrilateral elements, or eight subelements in the case of hexahedral elements. The term *leaf elements* is used to denote the set of all elements in the mesh that do not have subelements, hence, this set contains unrefined base elements, subelements on intermediate refinement levels that have no children and subelements belonging to the highest refinement level.

As mentioned in the previous section, certain topological components in a multi-level *hp*-mesh need to be deactivated to enforce mesh compatibility and linear independence of the basis functions as shown in Figure 3.2. It should be noted that only degrees of freedom associated with active topological components are considered when setting up the linear system of equations in a simulation. Furthermore, when a multi-level *hp*-mesh is used in combination with FCM, a further distinction has to be made between active and inactive (leaf) elements on the basis of their intersection with the physical domain Ω_{phys}. Elements completely outside Ω_{phys} are not taken into account in numerical simulations.

3.2.3 Numerical integration

An important aspect in *hp*-schemes based on the superposition principle is the correct numerical integration of the variational form. Standard elementwise quadrature rules cannot be directly applied to base elements in a multi-level *hp*-mesh since basis functions within these

elements are only piecewise polynomials. Instead, integration points have to be distributed with respect to the leaf elements as the basis functions in these elements are polynomial and thus C^∞-continuous.

Figure 3.3 depicts the integration domains on a single one-dimensional element with three levels of overlay elements. The base element is partitioned into four integration domains in which the basis functions are polynomial and only the basis functions supported on an integration domain are taken into account when evaluating quantities for a given subdomain. It should be noted that more elaborate integration strategies may be required when combining FCM and multi-level *hp*-refinement. In this case, any cut-cell integration strategy (see Section 2.3.2.1) can be applied on the leaf element integration domains, e.g. voxel or space-tree partitioning.

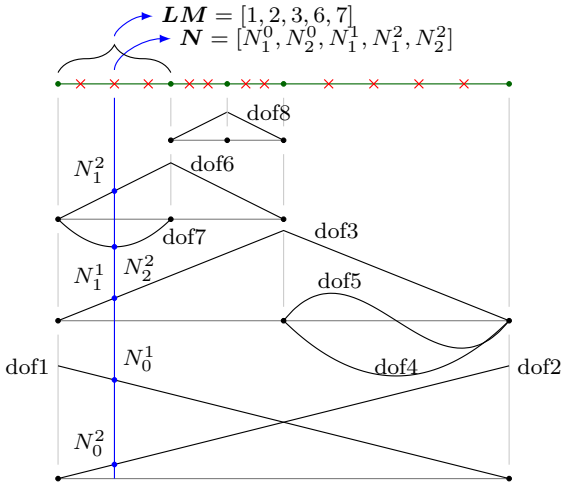


Figure 3.3: Illustration of Gaussian integration for a one-dimensional multi-level *hp*-refined mesh, adopted from [Zander et al., 2016a]. The integration domains are chosen such that the basis functions are C^∞-continuous within each domain.

3.3 A software framework for *hp*-refined high-order finite elements

All algorithms and numerical examples in this thesis have been realized within the software framework `AdhoC++`. The following section gives a short overview of the implementation and design of the serial version of the code framework. The interested reader is referred to [Zander, 2017; Zander et al., 2016a, 2015] for more elaborate information on the data structures and algorithms needed to enforce linear independence and compatibility of the multi-level basis. The extension of this code framework to allow for distributed-memory computations is the subject of Chapter 4.

3.3.1 Code structure and serial implementation

`AdhoC++` is an object-oriented finite element code developed by the Chair for Computation in Engineering at the Technical University Munich for performing numerical simulations with

high-order finite elements in the field of solid mechanics. The code's name is an acronym that stands for *Advanced high-order finite element Code* in `C++`. The programming language `C++` was chosen due to its efficiency, portability, scalability and support of object-oriented programming. Furthermore, several optimized libraries written in `C++` exist that can be used as a basis for building performant software. `AdhoC++` was originally designed as a shared-memory code in 2012 and supports both boundary-conforming finite element modeling and finite cell computations using quadrilateral and hexahedral elements with arbitrarily curved edges and faces. The framework mainly uses high-order elements based on a (isotropic or anisotropic) tensor product of univariate integrated Legendre polynomials, see Section 2.2.2. A core feature of `AdhoC++` is the implementation of the multi-level *hp*-refinement scheme. Classical *hp*-formulations generally require complex algorithms and data structures to consolidate and constrain hanging nodes and are associated with a high implementational effort. The natural treatment of hanging nodes in the multi-level *hp*-scheme, allows it to be implemented in a simple and intuitive manner using object-oriented programming concepts [Zander et al., 2015].

An important aspect in the design of finite element software is the organization of different functionality into libraries and modules. This encourages code reuse and reduces the effort associated with code extension and modification. Each library contains a set of algorithms and classes that are related to a specific aspect of the finite element pipeline e.g. the `geokernel` library is responsible for the creation and manipulation of geometric entities such as vertices and lines while the `fekernel` library handles aspects relating to the finite element spaces. The object-oriented concept also allows different relations to be defined between class objects using `C++` pointers. This is illustrated in the UML diagram in Figure 3.4, which shows three central interclass relations needed to perform refinement by superposition on high-order finite elements. These relations are *i)* each degree of freedom is uniquely associated with a topological entity, *ii)* a topological entity stores pointers to its subcomponents and *iii)* each element stores a pointer to its topological entities and pointers to its children.

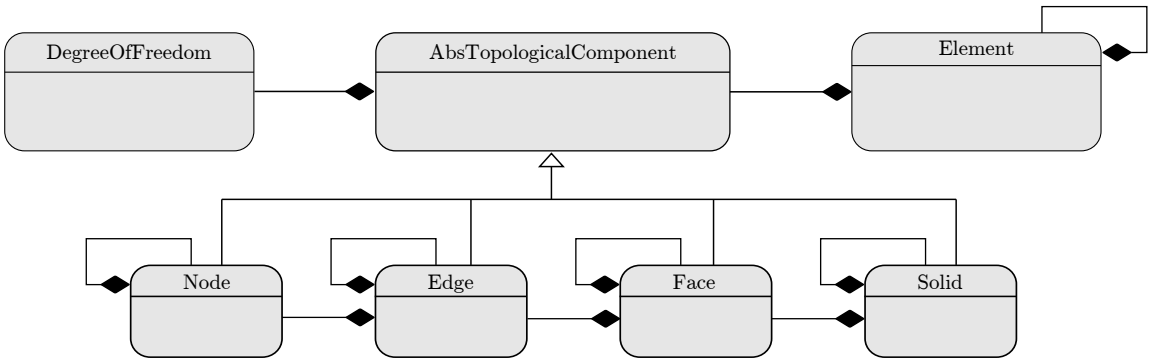


Figure 3.4: UML diagram adopted from [Zander et al., 2016b] depicting the relations between the DOFs, topological entities and elements. Expression of object relations using pointers allows an intuitive implementation of the multi-level *hp*-scheme.

Finite cell simulation pipeline

In order to perform a simulation in AdhoC++, one first needs to formulate an initial boundary value problem in a C++ file that is then compiled into an executable. Setting up a typical quasi-static finite cell computation in AdhoC++ involves the definition of the physical domain Ω_{phys}, which can be defined in several ways e.g. as an implicit function, a constructive solid geometry object or read in as input from stereolithography file (STL file) or a computer tomography image (CT image). Next, the geometric properties of the embedding mesh such as its spatial dimensions and element resolution need to be specified. Thereafter, the problem type and corresponding material properties, as well as the type of ansatz space that should be used is specified. If needed, a *hp*-refinement strategy for performing mesh adaptation can also be defined. The final stage in the simulation setup involves the specification of the boundary conditions that need to be applied, the choice of the linear system solver and the quantities that should be post-processed.

The stages in a finite cell computation do not differ much from those of a standard FE computation and include pre-processing, the integration of element matrices and their assembly, the solving of the resulting equation system and post-processing. In FCM simulations, integration is usually performed using one of the specialized integration techniques described in Section 2.3.2.1. During this phase, each integration point needs to be associated with the correct value of the indicator function and material properties through a simple query that depends on the geometrical model defining Ω_{phys}.

3.3.2 Code performance, bottlenecks and limitations

AdhoC++ has undergone several optimizations to improve its node-level performance. An important ingredient for improving single processor performance on modern computing architectures is the use of (Single Instruction Multiple Data) SIMD instructions, such as vectorization. Vectorization is extensively used in the code and significantly improves the performance when computing matrix-vector and matrix-matrix products during the integration and solution phases. Shared-memory multiprocessing through OpenMP directives is also heavily applied in the integration and solving phases.

One of the major bottlenecks of AdhoC++ is its high memory usage. This is attributed to the matrix-based solution approach adopted in the code and the memory footprint of the computational mesh. The multi-level refinement procedure implemented in AdhoC++ is based on pointers and requires that each topological entity be generated and stored as a distinct object in memory. The relation between different topological components, such as their adjacency, is also explicitly stored. This results in a rather large memory footprint associated with the mesh data structures.

Prior to the work done in this thesis, AdhoC++ had two major limitations that made the computation of large-scale finite cell problems involving more than a million elements and several million or even billions of DOFs unfeasible. Firstly, the code did not make use of distributed memory parallelism and could only be used on a single NUMA (non-uniform memory access) node. Secondly, at that time it was customary to use direct solvers to solve finite cell systems since research on iterative solution techniques for immersed problems was limited. The thesis at hand seeks to eliminate these restrictions and show how efficient FCM examples of industrial relevance can be computed using well-designed parallel data structures

and algorithms. Chapter 4 addresses how concepts of parallel computing can be incorporated in the code to enable the generation and manipulation of large FCM meshes on parallel sytems, while Chapter 5 presents robust iterative solvers for the efficient solution of large finite cell systems.

Chapter 4

Parallel immersed computations

The previous chapter presented the discretization methods that play a central role in this thesis, namely the finite cell method and the multi-level *hp*-scheme. The chapter at hand discusses the application of parallel computing to these methods and describes the algorithms and data structures needed for large-scale finite cell analyses in problems of engineering relevance. The content of this chapter is structured as follows: A short introduction to parallel computing will be given at the beginning that introduces the relevant parallel paradigms and the different metrics applied in this work to quantify parallel performance. Note that this introduction is by no means exhaustive and should be supplemented with secondary literature such as [Hager and Wellein, 2010] for readers unfamiliar with parallel algorithms in the context of scientific computing. Next, the ingredients needed for scalable parallel finite element computations will be highlighted and a review of different parallel frameworks for high-order finite elements is given. This will be followed by a presentation of the first attempt at a parallel implementation of `AdhoC++` that is based on replicated mesh data structures. The content in this section is based on the work reported in [Jomo et al., 2017]. In the final part of this chapter, a parallel framework for the finite cell method and multi-level *hp*-refinement based on an adaptive Cartesian mesh is presented. This approach allows for scalable computations on massively parallel systems like the SuperMUC-NG at the Leibniz Supercomputing Center in Munich, Germany.

4.1 Fundamentals of parallel computing

The term *parallel computing* is used to describe a group of computational units working simultaneously to solve a problem or a set of problems. For the sake of simplicity, a single core will be the smallest computing unit considered in this thesis and no details on parallelism involving its subcomponents such as vectorization or other forms of instruction-level parallelism will be discussed. It should be noted, however, that a code must exploit all levels of parallelism in order to achieve satisfactory performance. Modern computers ranging from a simple desktop to large computing clusters are equipped with multiple cores that provide several benefits, e.g. a user is able to run multiple programs simultaneously or experience an improved performance in a single application such as faster execution. Parallel computing in the field of scientific computing has two central goals, firstly it aims to accelerate computations and lead to lower execution times. The second goal is the use of combined resources to solve larger problems

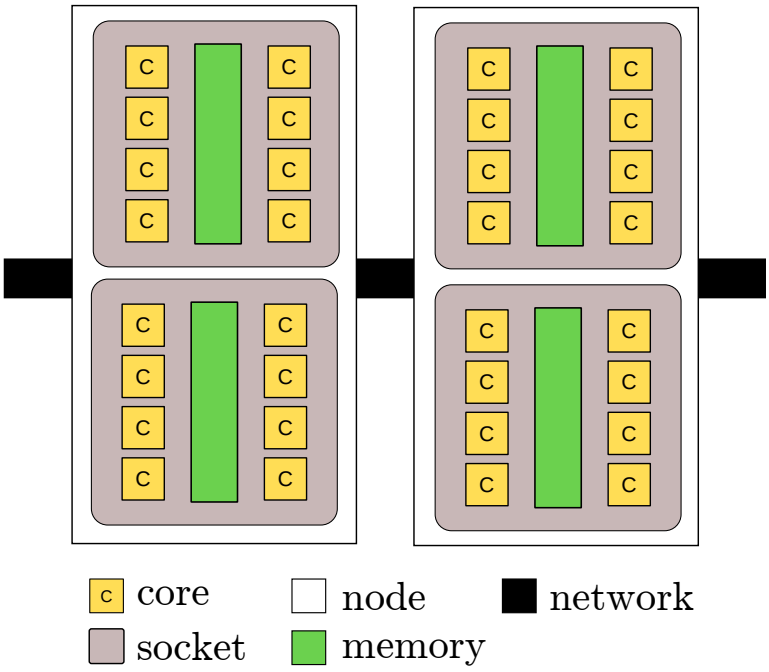


Figure 4.1: Simplified layout of a computing cluster depicting its various components.

than those that can be solved by a single core.

Before introducing the different types of parallelism, the hardware components used in high-performance computing are presented. Figure 4.1 shows a simplified layout of two computing nodes. Modern computing clusters are made up of several nodes that are connected with each other through a (high-speed) network. Each node usually contains a fixed number of physical cores (processors) that are grouped together into sockets. Each node also contains memory modules and other components that support the processing and transfer of information within the node.

Parallelism and parallel programming models

There are different forms of parallelism and only those that exploit multiple cores or nodes are considered in this work. Furthermore, only *data parallelism* is considered, where cores work concurrently to manipulate different subsets of the same data e.g. cores working in tandem to solve a linear system. This is in contrast to *task parallelism* where cores perform different independent operations on the same data. A parallel programming model is an abstraction that dictates how various components of a parallel machine can be used. It provides a generalization of how processors can interact and communicate with each other in order to formalize and simplify the design of parallel codes. The main parallel programming models commonly used in scientific codes are

- **Shared-memory models:** In these programming models, processors share a common physical address space (memory) and can generally only be used within a single node. They allow variables to be shared between processors and easily accessed during run-time without the need for communication between processors. Modification of shared

variables needs to be coordinated to ensure their validity. Prominent shared-memory programming models include *Open Multi-Processing* (OpenMP) [Dagum and Menon, 1998] and *Intel Thread Building Blocks* (TBB) [Reinders, 2007].

- **Distributed-memory models:** These programming models allow the use of several processors across multiple nodes. This is generally achieved through interprocessor communication by means of message passing over a communication network. A commonly used library that implements this programming model is the *Message Passing Interface* (MPI) [Clarke et al., 1994].
- **Hybrid models:** these models combine shared-memory parallelism within a node with a distributed-memory scheme such as MPI across nodes. They are often more sophisticated to program than pure MPI applications since different ways exist for partitioning processors between the two levels of parallelism. Hybrid models, however, reduce the number of messages required during program execution and allow the possibility of overlapping communication and computation that can be favorable in certain applications.

In this thesis, a hybrid programming model that utilizes OpenMP and MPI is applied. This approach is chosen for two main reasons. First, as previously mentioned, a hybrid approach can reduce parallel overhead as less MPI tasks participate in the communication during programm execution than in a pure MPI approach. Second, a hybrid code may require less memory than a pure MPI code in scenarios where the data that needs to be stored by a single MPI task does not scale well with the number of participating MPI tasks. Note that the terms MPI task or MPI process will be used interchangeably to refer to the compiled MPI application that is mapped to a processor when the code is executed. The term thread will be used to denote an OpenMP application that is created by the runtime environment and allocated to different processors at the beginning of an OpenMP region.

Quantifying parallel performance

One of the key goals of high-performance computing is the optimal use of the available computational resources i.e. cores and memory. For this reason, it is necessary to define different metrics that quantify parallel performance. These metrics simplify the identification of code bottlenecks and help guide code optimization. In the context of parallel algorithms, it is common to consider three metrics that are independent of the programming model used. These are *speedup*, *parallel scalability* and *parallel efficiency*. Speedup describes the ratio of the execution time of a program run in serial t_s, to that of the same program run with N processors t_N under the assumption that the amount of work remains the same i.e. speedup $= t_s/t_N$. Parallel scalability is closely related to speedup and is often used to judge the quality of a parallel algorithm. A differentiation is made between *strong scaling* that considers the speedup due to parallelism when the number of processors are increased for a problem of fixed size and *weak scaling* that describes the evolution of the execution time when the problem size is increased in proportion to the number of processors.

Under perfect conditions, an ideal strong scaling curve should be a linear curve with a slope of one. Likewise, perfect weak scalability translates to a constant execution time when the problem size is increased in proportion to the number of processors. Perfect scalability is, however, hard to achieve in practice due to several factors such as parallel overhead, load imbalance due to an uneven distribution of work between processors, the need for synchro-

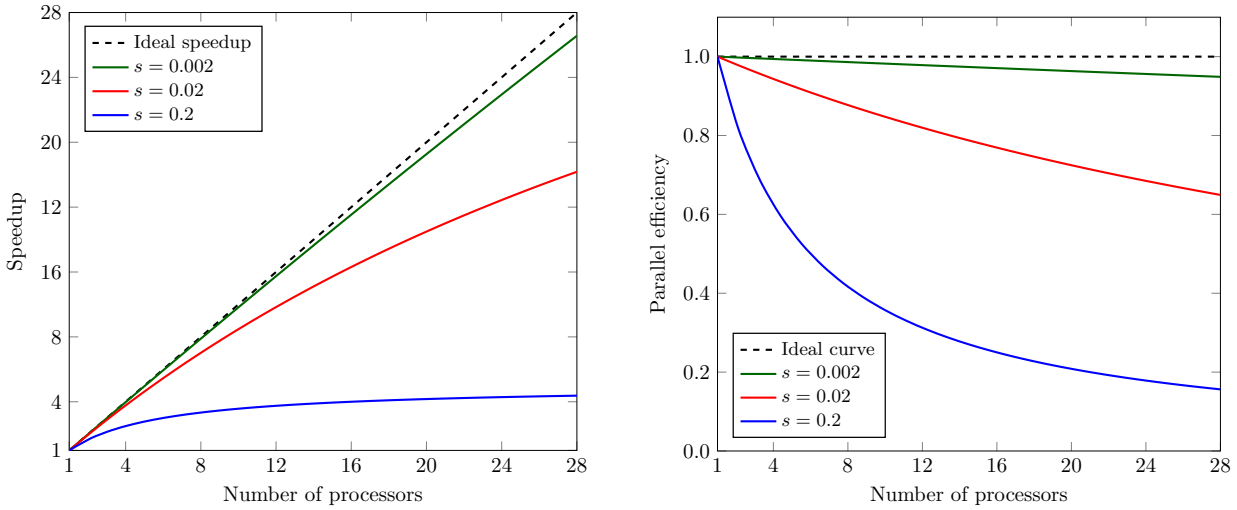
nization and the presence of serial parts of code [Hager and Wellein, 2010]. The effect of serial code segments can be taken into account when modeling the scalability one should expect. This is done through Amdahl's law [Amdahl, 1967] in the case of strong scaling and through Gustafson's law [Gustafson, 1988] in the case of weak scaling. These two relations provide a more realistic view of the speedup that should be expected in practice. Let s be the fraction of serial work in an application and p the fraction of parallelizable work such that $s + p = 1$, then the speedup in Amdahl's can be defined as

$$\textbf{Amdahl's law: } \text{speedup} = \frac{1}{s + \frac{p}{N}}. \tag{4.1}$$

It is also possible to provide a definition of speedup in the case of weak scaling as

$$\textbf{Gustafson's law: } \quad \text{speedup} = s + p \times N. \tag{4.2}$$

Parallel efficiency is usually defined as the ratio of the speedup to the number of processors N. Consequently, an application that makes optimal use of parallel resources has an efficiency of one. Figure 4.2 shows an example of speedup and efficiency curves for three different values of the serial work fraction computed using Amdahl's law.



(a) Speedup (b) Parallel efficiency

Figure 4.2: Exemplary curves showing the speedup and parallel efficiency for three values of the serial work fraction s computed using Amdahl's law.

4.2 Ingredients for scalable finite element analysis

Designing a parallel finite element framework with decent parallel scalability is an extremely involving task. It is of paramount importance that the chosen algorithms and data structures allow efficient and scalable use of the processors and available memory in order to facilitate optimal performance. Some of the best practices when designing parallel programs in a general setting are presented in [Foster, 1995] while aspects specific to parallel finite element code design are summarized in [Bangerth et al., 2011; Patra et al., 2003].

The parallelization of all stages of the finite element pipeline must be adequately addressed in order to yield an efficient code. In general, the following three ingredients are indispensable in a well-designed parallel code

- Scalable and efficient mesh management.
- Robust iterative solvers.
- Scalable post-processing.

4.2.1 Scalable and efficient mesh management

Mesh management in the context of parallel finite elements refers to all components and operations associated with the generation, storage and manipulation of mesh elements and their related data. The choice of data structures and algorithms to use for these operations is governed by factors such as the size of the target system that a parallel application will be run on, the type of computational mesh used i.e. whether it is structured or unstructured, and whether the mesh topology will be changing over time e.g. due to dynamic *hp*-refinements or domain growth as in the case of additive manufacturing processes. According to Patra et al. [2003] and Bangerth et al. [2011] a well-chosen mesh management strategy is characterized by *i)* distributed storage of the computational mesh, *ii)* fast algorithms for element retrieval, refinement and data exchange between processors and *iii)* efficient load balancing strategies that dynamically map mesh elements to different processors. Furthermore, point-to-point communications should be preferred over collective MPI calls and ought to be used in their place whenever possible.

There are several different strategies for representing and handling mesh data in parallel codes. The most commonly employed mesh management schemes can be grouped into

- **Mesh management schemes based on replicated resources:** In these schemes, the entire mesh data is generated and stored on each MPI task. Although such approaches inefficiently utilize available node memory, they are easy to implement and are often an initial step when transitioning from a serial code to a distributed-memory code. In fact many of the prominent scalable finite element codes currently available such as `deal.II` [Bangerth et al., 2007] and `libMesh` [Kirk et al., 2006] started out with replicated mesh data structures. Examples of finite element codes based on replicated meshes are [Jomo et al., 2017; Paszyński and Pardo, 2011].
- **Mesh management schemes based on distributed resources:** These schemes rely on the distributed storage of mesh data across all MPI tasks. They allow efficient usage of nodal memory and foster parallel scalability. Since no single MPI task knows the complete extent of the computational mesh, specialized algorithms are needed to determine a consistent degree of freedom numbering and maintain mesh compatibility across MPI tasks during simulations. A convenient way of incorporating distributed mesh functionality commonly used in many parallel codes is the use of packages such as `P4est` [Burstedde et al., 2011] or `ITAPS iMesh` [Ollivier-Gooch et al., 2010] that implement this functionality.

4.2.2 Robust and scalable solvers

Optimized parallel linear algebra routines and solvers are needed for achieving parallel scalability. Many parallel FE codes do not implement this functionality internally but rather leverage highly optimized implementations present in packages such as `Trilinos` [Heroux et al., 2005] and `PETSc` [Balay et al., 1997]. Trilinos for example provides a suite of classes for the construction of distributed sparse matrices, `Epetra` and `Tpetra`, that support distributed assembly of linear systems. It also contains an array of parallel iterative solvers in the packages `AztecOO` and `Belos`. Another mentionable parallel solver suite is the package `Hypre` [Falgout et al., 2006], which contains algebraic multigrid preconditioners and solvers for extreme-scale parallel systems. When deciding which parallel linear algebra and solver packages to use, one should have the following criteria in mind: *i)* The package should have well-designed interfaces that allow the use of user-defined routines. This is of particular importance in problems where custom preconditioning techniques are needed. *ii)* It is also beneficial to use packages with a large userbase and good support.

4.2.3 Scalable post-processing

The term post-processing is a very broad concept in the finite element method and refers to all operations performed once a finite element solution has been obtained. Such operations can be performed at the end of a simulation, between two consecutive simulations or when a simulation is still running. Typical post-processing operations in FEM can be grouped into three main classes *i)* the computation of secondary quantities derived from the FE solution e.g. the calculation of stresses within an element or a posteriori error estimates for the purpose of guiding adaptive mesh refinement, *ii)* the storage of simulation results to the file system and *iii)* the visualization of the simulation results for the purpose of drawing meaningful conclusions. Each of these operation classes is a vibrant research topic and cannot be thoroughly handled in this thesis. They, however, need to be sufficiently addressed to guarantee parallel scalability and efficient use of the parallel resources. The efficient writing and storage of data to the file system is a challenging task and falls under the research field of parallel I/O (Input/Output). Liu et al. [2010] elaborate on various approaches for massively parallel post-processing in the context of partitioned solver systems. Many parallel finite element codes leverage libraries such as HDF5 [The HDF Group, 1997] and Parallel NetCDF [Li et al., 2003] that provide standardized I/O operations. Commonly used applications for data analysis and parallel visualization include ParaView [Ahrens et al., 2005], Visit [Childs et al., 2012] and GiD [Coll et al., 2016].

4.2.4 A brief review of parallel frameworks for high-order finite elements

The use of parallel computing is not only restricted to research codes but is also present in many proprietary finite element software suites as well as several open-source finite element packages. A vast amount of parallel frameworks for FEA exist and an exhaustive review of these is beyond the scope of this work. In recent years, several community-based codes also known as general-purpose finite element codes have emerged. These codes have a wide userbase and implement generic functionality that can be used in different scientific and engineering

applications. Many of these codes have the capability of using different high-order finite elements and *hp*-methods. Noteworthy frameworks include `deal.II` [Bangerth et al., 2007], `libMesh` [Kirk et al., 2006], `DUNE` [Dedner et al., 2010], `FEMPAR` [Badia et al., 2018a] and `MFEM` [Anderson et al., 2020]. To the author's knowledge, they are only a few frameworks reported in literature that can perform parallel finite element computations using immersed methods. A parallel version of AgFEM is presented in [Verdugo et al., 2019] and is implemented in `FEMPAR`. This method has been extended to *h*-adaptive meshes in [Badia et al., 2020; Neiva and Badia, 2020]. A cutFEM code is used for parallel Fluid-Structure Interaction computations in [Schott, 2017].

4.3 A simple parallelization scheme based on replicated mesh data structures

This section is based on the work reported in [Jomo et al., 2017] and presents the first attempt at a parallel framework for multi-level *hp*-refinement and the finite cell method. The main motivation for this work was the development of a simple parallelization scheme that would tackle the major bottleneck in `AdhoC++` at the time, namely the integration of the element system matrices. For this reason, a hybrid framework (MPI+OpenMP) was designed based on parallel mesh management using replicated mesh data structures. A conceptually similar approach can be found in [Paszyński and Demkowicz, 2006]. The scheme proposed in this thesis partitions the active leaf elements are among MPI tasks and is briefly explained in the following section.

4.3.1 Parallel simulation pipeline

Algorithm 4.1 shows the simulation pipeline of the parallelization strategy based on a shared mesh data structure. Each MPI process generates the entire computational domain at the start of the simulation. Mesh refinement is performed on the entire mesh such that the same discretization is present on all MPI tasks. Next, the active leaf elements are partitioned among the processes using either a geometric or graph-based partitioner in `Zoltan` [Devine et al., 2002]. For partitioning purposes, each leaf element is associated with a weight w that is proportional to the amount of computational work related to it. Since the evaluation of the element matrices is usually the main bottleneck in high-order immersed computations, the weight of a given leaf element is chosen such that $w \approx n_{GP} * N^3$, where n_{GP} corresponds to the number of integration points and N the number of active basis functions supported on the leaf element. In concrete simulations, w is normalized with the weight of an unrefined element yielding a modified weight $w^* = w/w_0$, so as to factor in influences not taken into account during its derivation.

Once an MPI process has determined which leaf elements it is responsible for, it performs numerical integration on these elements. Shared-memory parallelism is used to accelerate this process and the results of the integration are stored in an intermediate linear system. The ownership of the degrees of freedom is determined just before the assembly of the distributed linear system. Since all MPI processes have knowledge of the complete computational domain, all DOFs already posses a consistent numbering. [Jomo et al., 2017] described two options for

determining the DOF ownership. Owned DOFs can be easily determined by either assigning blocks of contiguous DOFs to the MPI processes in an ascending order or assigning interface DOFs to the MPI process with the lowest rank. Once the DOF ownership has been determined, the global distributed system is assembled using the parallel matrices in `Trilinos` [Heroux et al., 2005]. These packages allow non-local matrix and vector entries to be communicated in an efficient way, which is essential in this scheme since ghost elements are not utilized. The distributed linear system is then solved with the parallel direct or iterative solvers in `Trilinos` and the post-processing carried out using the parallel hierarchical data format library HDF5.

Algorithm 4.1: Summary of the simulation pipeline

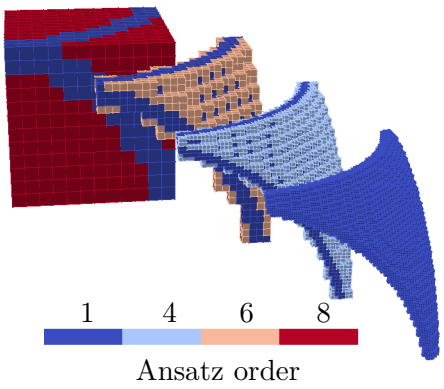
```
1  mesh = createMesh( )
2  for iStep : timeSteps do
3  |   refineElements( mesh )
4  |   myProcessLeafIds = getActiveLeafElementsOnProcess( mesh )
5  |   registerActiveLeafElementsOnProcess( myProcessLeafIds )
6  |   for iLeafElement : myProcessLeafElements do
7  |   |   #pragma omp for
8  |   |   for intPoint : Integrationpoints do
9  |   |   |   elementMatrices =+ calculateLeafElementMatrices( )
10 |   |   end
11 |   |   intermediateLinearSystem.scatterInto( elementMatrices )
12 |   end
13 |   myProcessDofs = distributeDegreesOfFreedom( mesh, myProcessLeafIds )
14 |   globalLinearSystem = initializeGlobalLinearSystem( myProcesDofs )
15 |   globalLinearSystem.assembleLinearSystem( intermediateLinearSystem )
16 |   solution = globalLinearSystem.solve( )
17 |   postProcessResults( mesh, solution )
18 end
```

4.3.2 Numerical examples

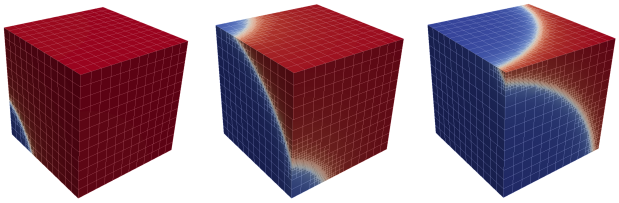
The following numerical examples showcase the performance of the proposed parallelization scheme that utilizes replicated mesh data structures. These simulations were computed on the CoolMAC cluster at Technical University of Munich that was equipped with a node architecture comprising 16 Intel Sandy Bridge-EP Xeon E5-2670 processors and 128GB of memory. This cluster is unfortunately no longer in service. All examples are solved using the parallel Conjugate Gradient solver with a multigrid preconditioner available in `Hypre`. Although the implementation can be run within a hybrid framework as portrayed in Algorithm 4.1, the numerical examples present only investigate the MPI-flat (only MPI) performance of the code.

4.3.2.1 A 3D Poisson problem involving complex refinement patterns

The first example investigates the scalability of our parallel implementation in a transient three dimensional Poisson problem involving complex mesh refinement patterns. This benchmark is

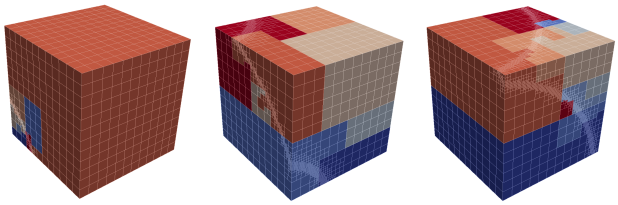


(g) Graded mesh at t=5.



(a) $t = 2$ (b) $t = 4$ (c) $t = 5$

Solution of the shock problem.



(d) $t = 2$ (e) $t = 4$ (f) $t = 5$

Distribution of the domain among 8 processes.

Figure 4.3: Problem setup of the transient shock problem.

known as the "shock problem". It is considered in [Rachowicz et al., 2006; Zander et al., 2016a] and entails solving the Poisson equation on a unit cube subjected to Dirichlet and Neumann boundary conditions derived from the manufactured solution u_{ref} that is specially chosen to represent a shock-like function

$$u_{\text{ref}} = \tan^{-1}(\tilde{\alpha}(r - r_0)) \quad \text{with} \quad r_0 = \sqrt{3} \quad \text{and} \quad \tilde{\alpha} \in \{40, 80, 160\}, \tag{4.3}$$

with r the radial coordinate about a shifted origin where

$$r = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} \quad \text{and} \quad x_0 = y_0 = z_0 = -0.25. \tag{4.4}$$

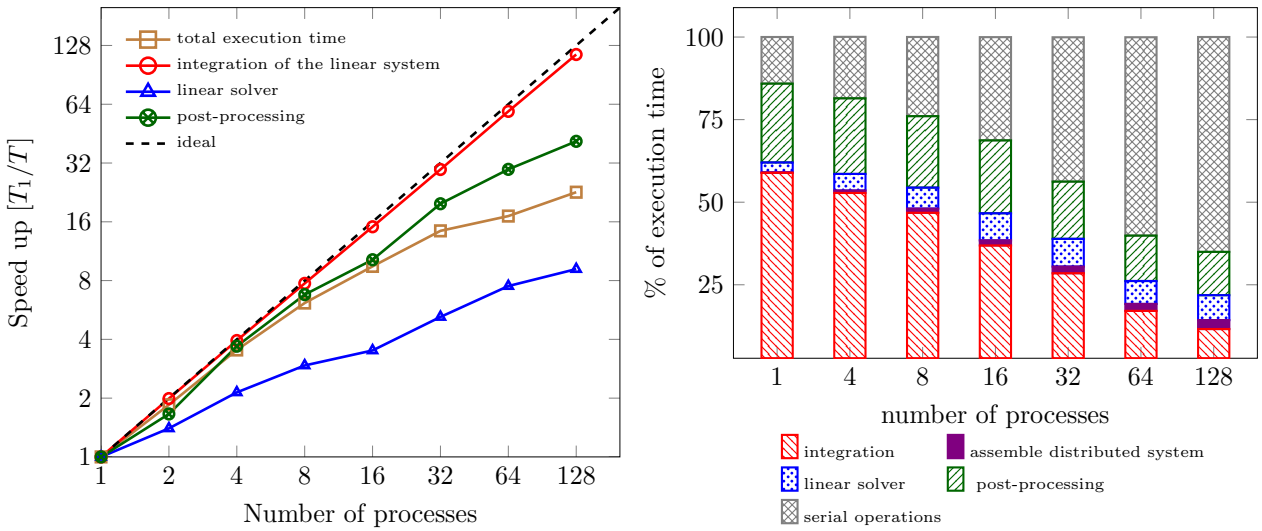
The factor $\tilde{\alpha}$ determines the sharpness of the shock. The interested reader is referred to [Zander et al., 2016a] for a detailed derivation of the applied boundary conditions and multi-level *hp*-convergence properties. In this numerical example, the original shock problem is modified to allow for a time-dependent initial radius $r_0 = r_0(t)$ in (4.3). The performance of the parallel framework in a transient setup is then assessed for a sharpness $\alpha = 160$ and different values of r as illustrated in Figure 4.3a-c. The example at hand begins in the first time step with an initial mesh of 12^3 hexahedral trunk space elements. The initial mesh consists of unrefined elements and has a uniform polynomial degree $p = 8$. This mesh is refined in every time step towards the shock with a refinement depth of three as illustrated in Figure 4.3d-f and a total of five time steps are computed. A graded polynomial order is applied to the elements, *cf.* [Zander et al., 2016a], as illustrated in Figure 4.3g for the final time step, where the polynomial order is decreased when increasing the refinement level. A graded mesh poses a challenge for load balancing, as the number of basis functions within the leaf elements varies greatly. Table 4.1 summarizes the number of leaf elements and unknowns within each time step.

Time step	1	2	3	4	5
r_0	$0.2\sqrt{3}$	$0.4\sqrt{3}$	$0.6\sqrt{3}$	$0.8\sqrt{3}$	$\sqrt{3}$
No. degrees of freedom	158 437	191 329	240 028	266 506	214 576
No. leaf elements	6 474	19 095	38 604	48 390	27 201

Table 4.1: Time step information in the transient shock problem.

Scalability of the simulation pipeline

To investigate the scalability of the parallel implementation, the time spent in performance-critical routines over the five time steps and their contribution to the overall simulation time is monitored. A strong scaling analysis is performed in which the processor count is progressively doubled in 7 steps from 1 to 128. Figure 4.4 presents the scaling results of different routines in the simulation pipeline as well as their influence on the execution time. Almost perfect scalability is achieved for the integration routines which make up over 50% of the serial computation time. The PCG solver does not scale well in this simulation due to the fact that rather small systems (with $\approx 2 \cdot 10^5$ DOFs) are considered in this study. These systems do to provide enough workload for the participating processors resulting in a high communication overhead in the linear solver. Increasing the processor count above 128 is likely to cause the performance to plateau or even decline due to increased parallel overhead. Figure 4.4b shows that the serial routines of the code, which include the mesh generation and refinement, have a detrimental impact on overall code performance. These routines do not scale and make up a major part of the computational time many MPI tasks are used.



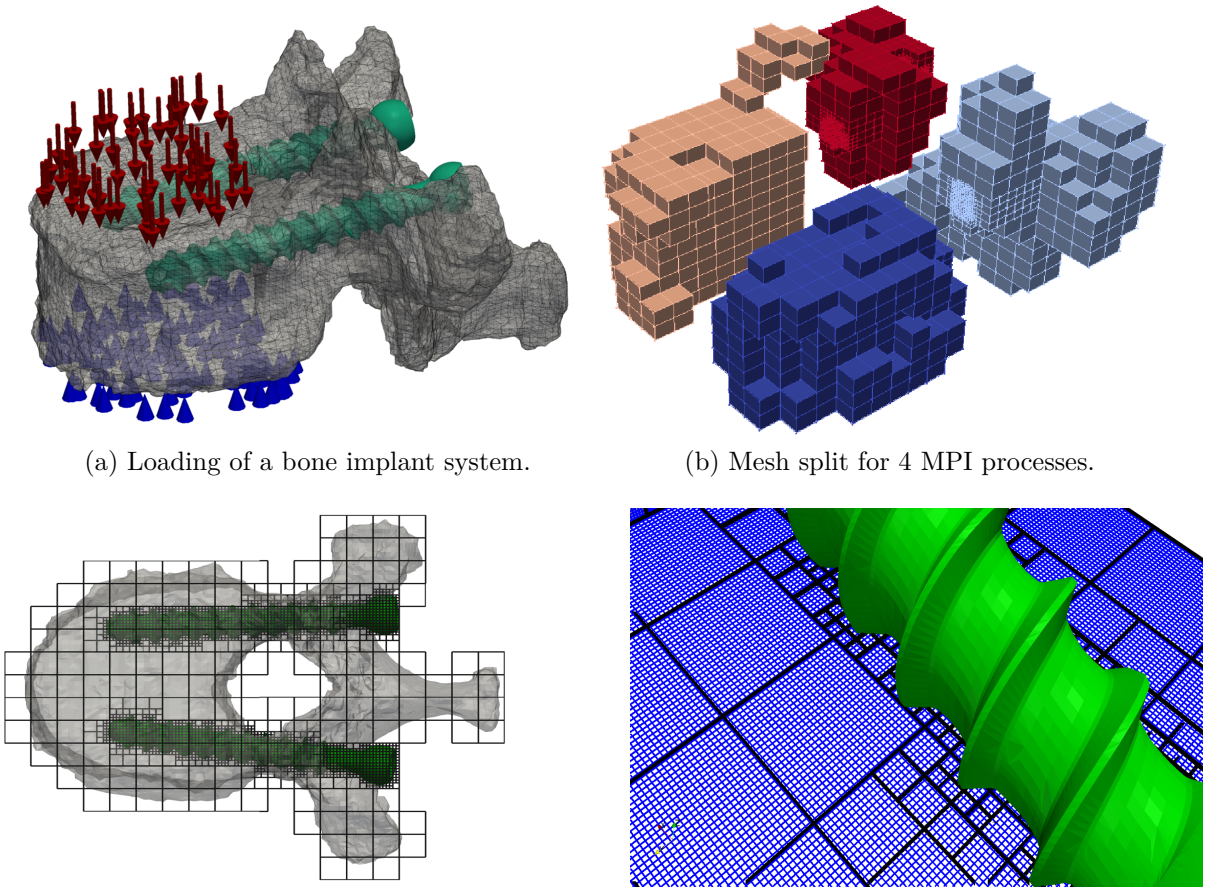
(a) Strong scaling results of selected routines.

(b) Contribution of different components to the total execution time.

Figure 4.4: Analysis of the overall performance in the shock problem.

4.3.2.2 Loading of a bone implant system

The final example in this section combines the finite cell method and multi-level *hp*-refinement to simulate the loading of a bone-implant system. The setup consists of a lumbar vertebra with embedded pedicle screws that is supported on its bottom surface and loaded on its upper flank as shown in Figure 4.5a. The major benefit of using FCM in this simulation is its ability to deal with multiple geometric models easily without the need of generating a boundary conforming mesh. This enables the simultaneous analysis on the bone material, which is based on a HR-pQCT scan with a voxel size of 146.5 μm, and the screws, whose geometry is described by a CAD model. A computational mesh shown in Figure 4.5b for example, can be easily generated by embedding both the voxel model and screw in a bounding box and using the density values of the CT scan together with the surface of the screws to filter out elements lying completely in the fictitious domain. The resulting computational domain is refined towards the interface of the bone material and the screws as illustrated in Figure 4.5c, to better capture the solution in this area. Furthermore, the composed integration technique in Section 2.3.2.1 is applied on the voxel level, to make use of the available fine grain information of the CT scan. This high resolution integration is depicted in Figure 4.5d, where the black colored cells represent the leaf elements while the blue colored cells depict the voxels. This results in a very high computational cost, rendering the integration of the linear system the main bottleneck in this simulation.



(a) Loading of a bone implant system.

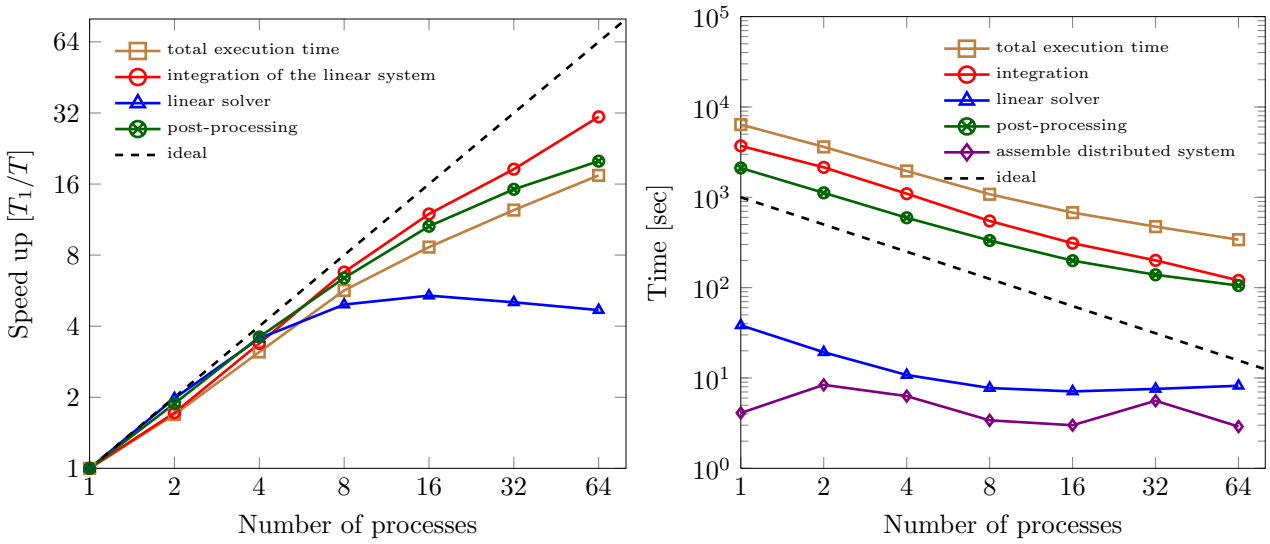
(b) Mesh split for 4 MPI processes.

(c) Adaptive refinement.

(d) Composed integration on voxel level.

Figure 4.5: Simulation of a bone-implant system.

This example presents a excellent test for the parallel implementation. The major challenge here lies in the efficient distribution of the computational domain to guarantee good performance. The leaf elements not only differ in the number of DOFs due to refinement, but also in the number of voxels. Moreover, an inside-outside test has to be performed for each integration point to find out if the point lies within the bone material, screw or fictitious domain. To perform the numerical analysis, an initial mesh comprising 1 341 base elements that are refined in two steps towards the interface of the bone and the screws is used. A polynomial degree of $p = 3$ is chosen resulting in 7 571 leaf elements upon refinement and 147 889 DOFs.



(a) Strong scaling results for selected routines.

(b) Execution time analysis of the simulation pipeline.

Figure 4.6: Analysis of the bone-implant system simulation.

Figure 4.6 shows the parallel performance of the simulation pipeline for the example at hand. Satisfactory scaling results are obtained for the most time consuming routines: the integration of the stiffness matrices and the post processing. More effort, however, needs to be invested in load balancing, i.e. in the distribution of the integration domains among processes, in order to improve the algorithms parallel efficiency.

4.3.3 Limitations of the parallel implementation

Although the proposed parallelization scheme is able to yield satisfactory results for problems with a few hundred elements per process, its applicability is limited due to its high memory requirements. A study of the code's memory footprint is conducted in [Jomo et al., 2017] and shows that the memory usage does not scale well when the number of MPI processes used in a simulation increases, but rather rapidly levels off due to the replicated mesh. This scheme nevertheless shows the potential of using parallel computing in the context of the finite cell method together with multi-level *hp*-refinement and serves as a basis for developing the more efficient parallel code described in Section 4.4.

4.4 A massively parallel framework for finite cell analysis

This section presents a distributed mesh management strategy that was designed to overcome the shortcomings of the shared mesh strategy outlined in the previous section. The core idea behind the proposed scheme is the introduction of an adaptive Cartesian grid that is easy to construct, light-weight i.e. has low storage requirements, and is easy to modify in that the cells it contains can be easily refined and coarsened. This grid provides an abstraction of the mesh in the sense that geometrical operations and load balancing can first be performed on its cells, before being related to a finite cell mesh. Once a distributed grid has been constructed, each MPI task is able to generate a local mesh from the grid data structure that can be used for parallel FE computations. In this way, it is possible to create and store a full parallel mesh in a scalable manner.

As previously mentioned, the concept of using adaptive Cartesian grids or tree-based structures for parallel element creation is not new but has been implemented in various parallel finite element codes e.g. [Badia et al., 2019; Bangerth et al., 2011]. The mesh management strategy proposed in this thesis is specially tailored to the needs of the finite cell method and multi-level *hp*-refinement and allows efficient immersed computations involving multi-million and even billions of unknowns on over 10K processors. The presented algorithms can, however, be easily extended to other immersed methods involving *hp*-refinement. This section presents a set of algorithms for the generation of a fully-distributed, analysis-suitable FCM mesh as well as methods for mapping element data between MPI processes in the case of parallel simulations involving changing discretizations. Before explaining the algorithms in detail, the terminology and notation used in this section are introduced.

4.4.1 Parallel mesh generation

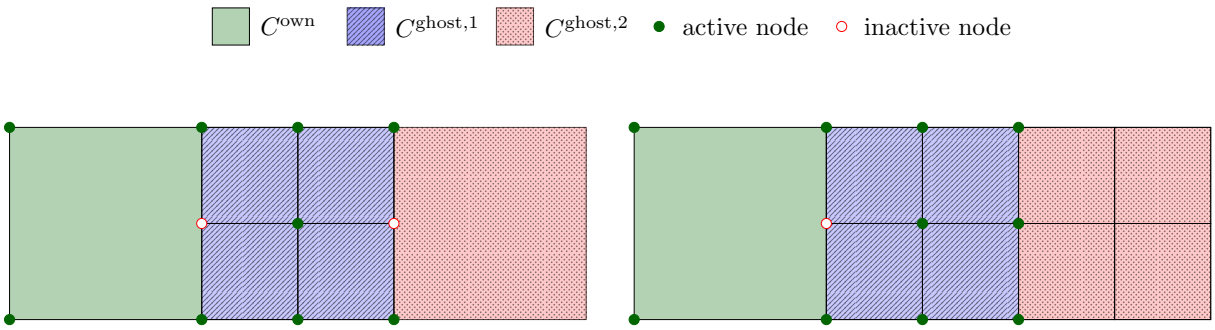
Terminology, notation and classification of the grid cells

Let $\mathcal{G}$ denote an adaptive Cartesian grid consisting of grid cells, with C denoting a single grid cell. Each cell can be uniquely identified by its index j and refinement level l. $\mathcal{G}_{\text{init}}$ denotes the initial state of the Cartesian grid in which all cells have a refinement level $l = 0$. This grid will be referred to as the coarse initial grid. Refinement of cells in the grid is performed using uniform bisections resulting in four and eight child grids in two and three dimensions respectively. When dealing with adaptive grids in a parallel setting, it is common to group the grid cells into different categories. These categories play a key role in the mesh manipulation algorithms and are listed in the following paragraph:

- **Inside and outside cells**: In analogy to inside and outside elements, a differentiation is made between grid cells that have an intersection with the physical domain and those that are completely within the fictitious domain. C^{in} denotes an inside cell while C^{out} a cell with $C \cap \Omega_{\text{phys}} = \emptyset$. The set of all inside cells is denoted by $\mathcal{C}^{\text{in}}$, while $\mathcal{C}^{\text{out}}$ represents the collection of all outside cells.
- **Owned, and ghost cells**: Since the adaptive Cartesian grid is created and stored in a distributed manner, it is necessary to associate each grid cell that will be used for element generation with a unique owning MPI task. The ownership of the grid

cells is determined during the load balancing process performed using `Zoltan`. The symbol C^{own} is used to represent a single owned cell and $\mathcal{C}^{\text{own}}$ the set of cells owned by a specific MPI task. Cells neighboring (or adjacent to) owned cells are referred to as *ghost cells*. The presented algorithms require two levels of adjacent cells and differentiate between immediate neighbors of owned cells called first-level ghost cells and second-level ghost cells. $C^{\text{ghost},1}$ denotes a first-level ghost cell and $C^{\text{ghost},2}$ a second-level ghost cell whereas $\mathcal{C}^{\text{ghost},1}$ and $\mathcal{C}^{\text{ghost},2}$ represent all first-level and second-level cells in $\mathcal{G}$, respectively. The second-level of ghost cells is needed for three reasons: *i)* it allows the direct communication of degrees of freedom between processes in a single communication step as described in Section 4.4.2, *ii)* it ensures that the refinement of all first-level ghost elements is consistent as illustrated in Figure 4.7 and *iii)* it enables the preconditioners to be constructed without communication as elaborated in Section 5.2.4.3.

- **Active and inactive cells**: This label is used on MPI task level to indicate which cells should be considered during the mesh generation process. It follows from the previous grid cell categories that only inside cells that are either owned or ghost cells can be active on a given MPI task. An active cell is denoted by C^{act} while C^{inact} denotes an inactive cell. The set of all active and inactive cells are denoted by $\mathcal{C}^{\text{act}}$ and $\mathcal{C}^{\text{inact}}$, respectively.



(a) Scenario 1: $C^{\text{ghost},2}$ is unrefined.

(a) Scenario 2: $C^{\text{ghost},2}$ is refined.

Figure 4.7: A simple grid consisting of three elements (with $p = 1$) that illustrates the importance of second-level ghost elements in ensuring the validity of DOFs on the interface between $\mathcal{C}^{\text{ghost},1}$ and $\mathcal{C}^{\text{ghost},2}$ when using multi-level *hp*-refinement. In the figure at hand, the refinement state of $\mathcal{C}^{\text{ghost},2}$ determines whether the node lying between $\mathcal{C}^{\text{ghost},1}$ and $\mathcal{C}^{\text{ghost},2}$ is active or inactive. Note that the DOFs in $\mathcal{C}^{\text{ghost},2}$ do not play a role in any of the algorithms used in this work and thus their correct state does not need to be enforced.

Mesh creation algorithm

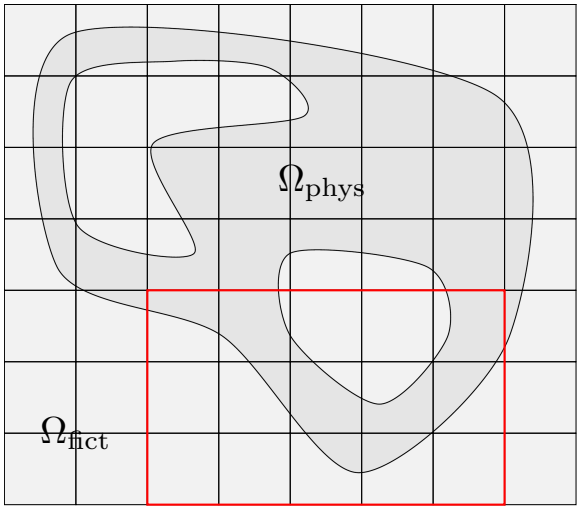
The algorithm proposed in this work generates a distributed finite cell mesh $\mathcal{T}$ from a light-weight adaptive Cartesian grid $\mathcal{G}$ that is structurally similar to the computational mesh. Each MPI task only stores a portion of the Cartesian grid denoted by $\mathcal{G}_{\text{rank}}$ and is able to generate a local high-order mesh $\mathcal{T}_{\text{rank}}$ from the active cells in $\mathcal{G}_{\text{rank}}$ at a given refinement level l_m. The ability to specify the grid refinement level for element generation is advantageous in that it allows the creation of a high-resolution FCM mesh from a rather coarse initial grid. Moreover,

a grid refinement level l_r and a mesh creation level l_m can be specified at the start of the mesh creation procedure making it possible to generate a uniform FCM mesh when $l_r = l_m$ and a multi-level *hp*-refined mesh for $l_r > l_m$. A 1-to-1 relation always exists between the generated elements and the active grid cells in $\mathcal{G}^{\text{rank}}$ with $l \geq l_m$ (see Figure 4.8). The full computational mesh is obtained from the union of all local MPI task meshes i.e. $\mathcal{T} = \bigcup_i \mathcal{T}_i$. A schematic illustration of the parallel mesh generation process is provided in Figure 4.8 while a summary of the scheme is presented in Algorithm 4.2.

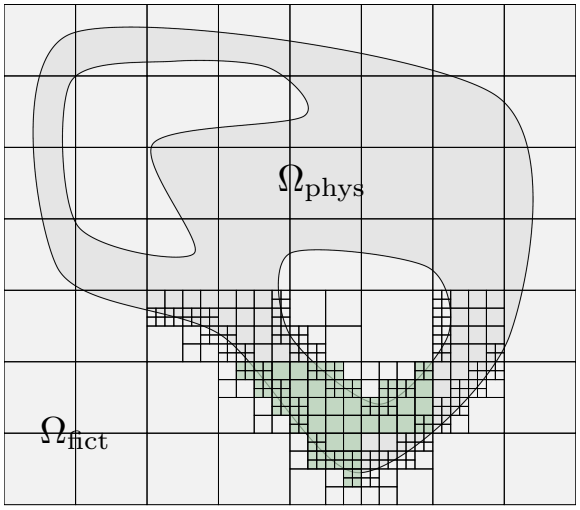
The mesh generation algorithm consists of the following steps:

1. An initial coarse grid $\mathcal{G}_{\text{init}}$ that can be readily stored on each MPI task is created, see Figure 4.8a.
2. An inside-outside test is performed to determine the set of inside cells $\mathcal{C}^{\text{in}}$.
3. The cells in $\mathcal{C}^{\text{in}}$ are partitioned among the MPI tasks using `Zoltan` [Devine et al., 2002]. The space-filling curves available in `Zoltan` are used to this end. Each process thereafter marks its active cells $\mathcal{C}^{\text{act}}$, with $\mathcal{C}^{\text{act}} = \mathcal{C}^{\text{own}} \cup \mathcal{C}^{\text{ghost},1} \cup \mathcal{C}^{\text{ghost},2}$.
4. Each MPI task refines the cells in $\mathcal{C}_{\text{act}}$ recursively until a predefined grid refinement level l_r, see Figure 4.8b.
5. The active cells with a refinement level $l = l_m$ are repartitioned using `Zoltan`. This step is necessary since the refinement process generally leads to an imbalance in the number of active cells that needs to be remedied, compare Figure 4.8b and Figure 4.8c.
6. Each MPI task then determines the new sets of ghost elements $\mathcal{C}^{\text{ghost},1}$ and $\mathcal{C}^{\text{ghost},2}$ and updates the refinement state and owning MPI process of these ghost cells. At the end of this step, each MPI task knows the owning MPI processes of all its active cells. This information is required to determine the point-to-point interprocess communications patterns needed in subsequent stages of the simulation pipeline.
7. A high-order finite cell mesh $\mathcal{T}_{\text{rank}}$ is then generated from all the active cells in $\mathcal{G}_{\text{rank}}$ with $l \geq l_m$, see Figure 4.8d.

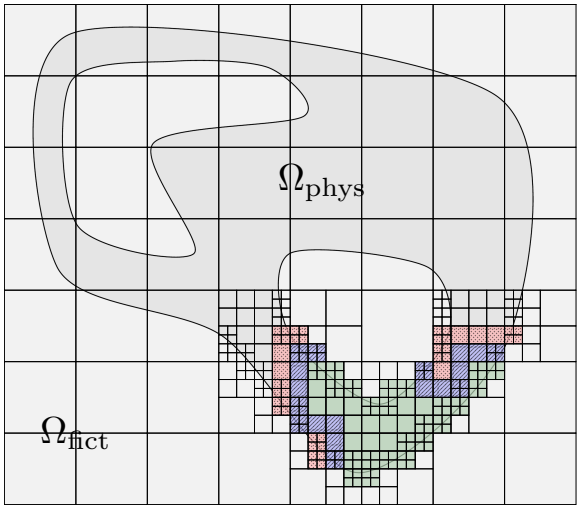
Remark 4.4.1. *Load balancing in the proposed parallelization scheme is performed on the granularity of the cells in $\mathcal{G}$ that correspond to the base elements of the multi-level hp-refined grids created on each MPI process. The computational cost of each base element, or its weight w, is chosen in proportion to the number of leaf elements it contains as well as the number of integration points on each leaf element.*



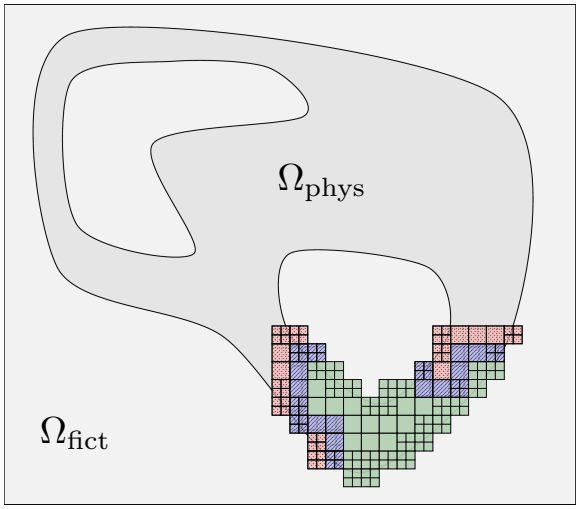
(a) An initial coarse grid $\mathcal{G}_{\text{init}}$ is generated and partitioned using `Zoltan`. The region enclosed in the red box depicts the initial subdomain $\mathcal{G}_{\text{init},1}$ assigned to the MPI task with rank = 1.



(b) The initial subdomain $\mathcal{G}_{\text{init}}$ is refined w.r.t the fictitious domain until a predefined depth l_r=3. The owned grid cells (green) are marked and again repartitioned to account for possible load imbalance introduced during the refinement process.



(c) Updated grid $\mathcal{G}_1$ after the load balancing of the refined grid cells with $l_m > 2$. Owned grid cells (green) and their neighbors (blue and red) are marked for the next stage of mesh generation.



(d) A refined FCM mesh $\mathcal{T}_1$ is generated from the active grid cells on MPI process 1.

Figure 4.8: Schematic representation of the generation of a *hp*-refined finite cell mesh from the point of view of a single MPI process (rank = 1).

Algorithm 4.2: $\mathcal{T}_{\text{rank}} = \text{generateMesh}(\ \Omega_{\text{phys}},\ \mathcal{G}_{\text{init}},\ l_r,\ l_m\ )$

```
# perform an inside-outside test to determine C_in
for C ∈ G_init do
    if C ∩ Ω_phys ≠ ∅ then
        C.setInsideState( true )
    end
end
# partition inside cells using Zoltan
C_own = partitionCells( G_init )
# update ghost cell states
G_rank = updateGhostCells( G_rank, C^own, l_m )
# refine locally active cells until depth l_r
for C^act ∈ G_rank do
    refineCellRecursively( C^act, Ω_phys, l_r )
end
# re-partition active cells with l = l_m using Zoltan
C^own = repartitionCells( G_rank, l_m )
# update ghost cell states
G_rank = updateGhostCells( G_rank, C^own, l_m )
# create a local mesh from active cells in G^rank
T_rank = extractMesh( G_rank, l_m, l_r )
return T_rank
```

4.4.2 Parallel enforcement of mesh compatibility

The process by which each MPI task generates a portion of the total computational mesh from the active cells in a distributed adaptive Cartesian grid was outlined in the previous section. It should be noted, however, that these meshes cannot be directly used for parallel simulations since the finite element spaces they define are independent of each other and first need to be connected. The process of connecting these local meshes will be referred to as the enforcement of parallel mesh compatibility. This procedure is necessary in order to obtain an *analysis-suitable discretization*. In the context of parallel finite cell computations, a computational mesh is considered analysis-suitable when the following criteria are met

(i) Each degree of freedom in the computational mesh $\mathcal{T}$ is associated with a unique global identifier (global index) and a unique owning MPI task.
(ii) The (point-to-point) interprocessor communication patterns needed to update data associated with the degrees of freedom of the ghost elements in $\mathcal{T}_{\text{rank}}$ have been established.

The enforcement of mesh compatibility results in a consistent enumeration of DOFs across all MPI tasks. With this in place, a distributed finite element space formed by the union of

all owned elements exists that has the correct inter-element continuity and total number of unknowns.

Degree of freedom nomenclature

In parallel finite element codes, it is common to decouple an MPI task's mesh from the routines that enforce mesh compatibility and instead use separate classes (or a single class) that implement(s) the logic of connecting the local finite element spaces across all processes e.g. [Bangerth et al., 2011; Dedner et al., 2010]. The mesh management strategy presented in this thesis is also based on a decoupled mesh approach and implements a class called the `DistributedDofHandler` that is responsible for managing the degree of freedom data and setting up the data structures needed to perform interprocessor communication.

Every DOF in an MPI task's local mesh is associated with a triple that contains the DOF's *local index*, *global index* and the *owning process* i.e. the rank of the MPI task that currently owns the DOF. The set of local DOF indices in the local mesh is denoted by $\mathcal{I}_{\text{local}} = \{i\}_{i=1}^{n_{\text{local}}}$, where n_{local} represents the number of DOFs in $\mathcal{T}_{\text{rank}}$. The symbol $\mathcal{I}_{\text{global}}$ is used to represent the set of global DOF indices in $\mathcal{T}_{rank}$. The values assigned to a DOF's global index and owning process are determined during the enforcement of mesh compatibility and are used to characterize the DOFs into the following groups

- **Valid vs invalid degrees of freedom:** The DOFs in $\mathcal{T}_{rank}$ are associated with three kinds of basis functions: *i)* basis functions that are fully supported on owned elements, *ii)* functions with support on both owned and ghost elements and *iii)* functions that are only supported on ghost elements. A DOF with a local index i is considered as valid on a given MPI task if the support of its corresponding basis function φ_i intersects at least one function φ_j that is fully supported on the union of owned elements in the local mesh. All other DOFs, for which this condition does not hold, are considered as invalid DOFs. The symbols $\mathcal{I}_{\text{valid}}$ and $\mathcal{I}_{\text{invalid}}$ represent the set of valid and invalid DOFs respectively. From the above definition, it is clear that invalid DOFs can only occur on second-level ghost elements, see Figure 4.9. An MPI task only needs to have the correct global indices of its valid DOFs since only these DOFs play a role in the assembly of the distributed linear system.
- **Owned vs remote degrees of freedom:** The valid DOFs of each MPI task can be further subdivided into two distinct sets, namely owned DOFs denoted by $\mathcal{I}_{\text{owned}}$ and remote DOFs denoted by $\mathcal{I}_{\text{remote}}$. An MPI task is responsible for the storage and manipulation of finite element data associated with its owned DOFs. This is of particular importance when coordinating the creation of parallel matrices and vectors or when performing linear algebra operations such as matrix-vector multiplications or the calculation of norms over distributed data structures. An MPI task can also store data associated to remote DOFs in $\mathcal{T}_{\text{rank}}$. In general, it is crucial that the remote data stored is valid i.e. data associated with a remote DOF k is consistent with the data on the MPI task that owns k. If this is not the case, then inconsistent data must be updated through interprocessor communication with the owning MPI task responsible for DOF k.

Remark 4.4.2. *It should be noted that each DOF in $\mathcal{T}$ is associated with only one owning MPI task and a unique global index. Moreover, the total number of degrees of freedom n_{DOFs} in $\mathcal{T}$ can be determined as the cardinality of the union of $\mathcal{I}_{\text{owned}}$ over all MPI tasks. Although*

trivial, it is worth mentioning that n_{DOFs} is an invariant that is independent of the number of MPI tasks used in a simulation or the distribution of the mesh elements over these tasks.

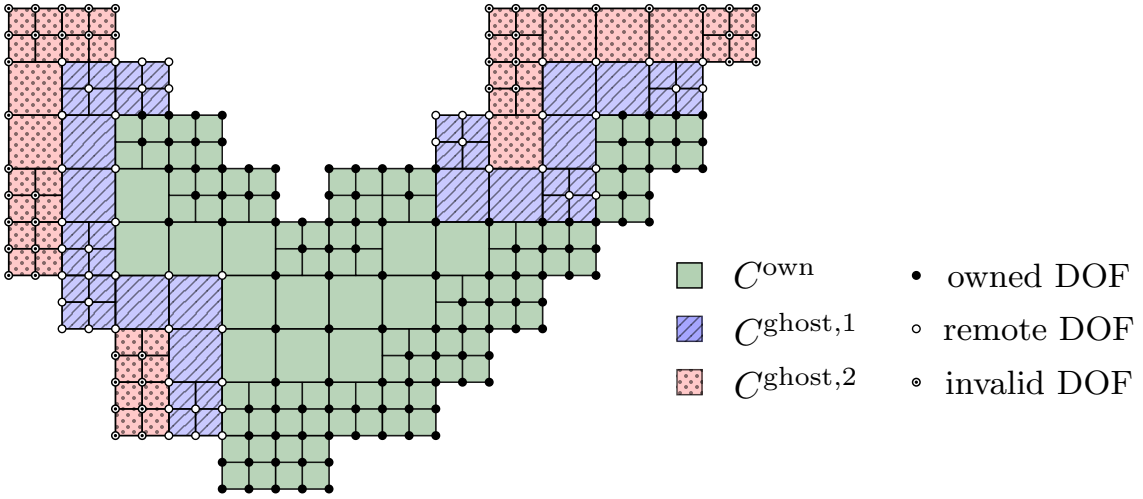


Figure 4.9: Classification of the degrees of freedom in the local mesh $\mathcal{T}_1$ with linear elements into invalid DOFs, remote DOFs and owned DOFs. $\mathcal{T}_1$ is the mesh belonging to the MPI process with rank one in Figure 4.8d.

Degree of freedom ownership

Before highlighting the specific steps needed to enforce mesh compatibility, we first elaborate on how the ownership of degrees of freedom is determined. This step is an integral part in parallel finite element computations. It must produce consistent results across all MPI tasks and should require minimal interprocessor communication [Bangerth et al., 2011]. When considering DOF ownership, one should distinguish between internal DOFs that are associated with basis functions only supported on owned elements in $\mathcal{T}_{\text{rank}}$ and interface or shared DOFs that are supported on owned and first-level ghost elements and whose corresponding basis functions are present on multiple MPI tasks. Since internal DOFs are only present on a single MPI task, it is a natural choice to assign this task as their owning MPI process.

Determining the ownership to interface DOFs is more involved. Different strategies exist in literature that differ in the rules used for assigning DOF ownership and the type and amount of interprocessor communication they require. One commonly used strategy is to assign the ownership of interface DOFs to the MPI task with the smallest rank that contains the given DOF as in [Bangerth et al., 2011; Logg, 2009]. An alternative strategy designed for nodal finite elements and octree-based meshes treats each nodal DOF as an octant that lies in a unique enclosing element and assigns the DOF to the MPI task that owns the enclosing element [Burstedde et al., 2011; Tu et al., 2005]. The algorithm presented in this thesis is also based on geometrical properties but follows a different approach. Since the basis functions and consequently the DOFs are associated with mesh topological entities – nodes, edges, faces, volumes — one can assign ownership of a DOF to the element with the lowest grid index that contains the DOFs topological entity. This element is referred to as the *anchor element* of the DOF in question. This approach is suitable for several reasons: firstly it is not restricted to hierarchical bases but can be readily applied to different kinds of finite element spaces.

Secondly, the algorithm does not require any communication since the index of any given element is equal to that of its corresponding grid cell and therefore unique and known a priori. Finally, this strategy simplifies the process of debugging since DOF ownership is associated with the geometry and independent of the partitioning of elements among MPI tasks. Figure 4.9 depicts the owned DOFs in a local mesh of a single MPI task.

Algorithm for enforcing parallel mesh compatibility

The process by which the finite element spaces defined by the local meshes on different processors are connected with each other is now described from the point of view of a single MPI task. This procedure involves the interplay of the local grid $\mathcal{G}_{\text{rank}}$, the local computational mesh $\mathcal{T}_{\text{rank}}$ and the `DistributedDofHandler` and consists of the following steps:

1. Determine the value of n_{local}, i.e. the number of DOFs in $\mathcal{T}_{\text{rank}}$, and initialize the local DOF indices with the values from 1 to n_{local}.
2. Distinguish between the valid and invalid DOFs based on the adjacency relations of the elements and the support of the basis functions in $\mathcal{T}_{\text{rank}}$. Mark the invalid DOFs as de-registered by assigning an invalid value of -1 to each invalid DOF's global index and owning process.
3. Determine the owned and remote DOFs in the set of valid DOFs and initialize the owning process of each owned DOF with the value rank. The owning processes of the remote DOFs are also initialized at this stage following the algorithm outlined in the previous section.
4. Once the set of owned DOFs in $\mathcal{T}_{\text{rank}}$ is established, the global index of all owned DOFs, as well as the total number of DOFs contained in $\mathcal{T}$, can be determined. To this end, an `MPI_Scan` operation is carried out to determine the start index that an MPI task should use to enumerate its owned DOFs. The global index of the i^{th} owned DOF in $\mathcal{T}_{\text{rank}}$ is calculated by offsetting i with the start index.
5. The final step in the enforcement of mesh compatibility is the communication of global indices between MPI tasks so that each MPI task is aware of the correct global indices of all its remote DOFs. To achieve this, a simple ruleset that allows an elegant communication algorithm is applied. In this scheme, a distinction is made between *send DOFs*, owned degrees of freedom that a processor sends to its neighbors, and *receive DOFs*, that constitute remote DOFs on first-level ghost elements whose indices and values need to be obtained from adjacent processes. The proposed algorithm has the following characteristics: *i)* Communication is performed in a single stage involving only non-blocking point-to-point communication between any two given processors. *ii)* Data is always sent directly from one MPI task to another and not via any other processors. iii) All operations are performed in parallel and communication takes place only during the data exchange phase. iv) The algorithm ensures correct global indices of all DOFs on first-level ghost cells and not only for DOFs on the interface of owned and first-level ghost elements as in [Logg, 2009]. v) No step in the algorithm needs to be repeated as in [Bangerth et al., 2011] and send data is clustered before sending it to the desired destination to avoid duplication of entries in the send buffer. The communication of DOF indices is achieved through the following procedure:

   (a) Identify the valid DOFs that will be received from other MPI tasks by looping over

all first-level ghost elements in $\mathcal{T}_{\text{rank}}$. Each receive DOF is associated with a single pair containing the DOF's local index and the rank of its owning MPI task. This pair is inserted into a list of pairs called the receive vector. Since a DOF is generally supported on multiple elements, duplicate entries exist in the receive vector. At the end of this step, the receive vector is sorted and duplicate entries are eliminated.

(b) Identify the valid DOFs that will be sent to other MPI tasks by looping over all owned elements that share a common mesh entity with any first-level ghost element $K^{\text{ghost},1}$. Each send DOF is associated with one or more pairs containing its local DOF index and the rank of the MPI task it should be sent to. These pairs are inserted in a list and duplicate entries contained in the list removed after sorting. At the end of this step, the number of pairs associated with a single send DOF corresponds to the number of MPI tasks that contain the given DOF in $\mathcal{I}_{\text{remote}}$.

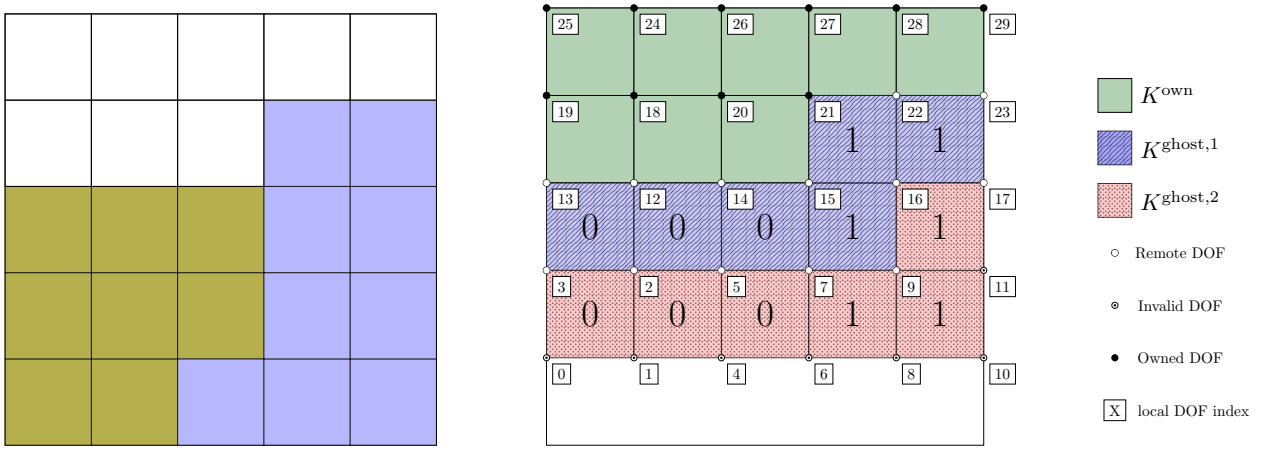
(c) Concatenate the entries in the send vector and receive vector that have the same destination or source in order to obtain a list of indices that should be sent to or received from a given processor. This step allows communication between any two given MPI tasks to be done in a single send and receive step.

(d) Create buffers where the global indices should be stored and initialize the send buffers with the correct global DOF indices.

(e) Perform non-blocking point-to-point communication between the MPI tasks. Note that the communication pattern between any two MPI tasks p_i and p_j is symmetric as they both send and receive data from each other.

(f) Read the data from the receive buffers and update the global indices of the remote DOFs in the `DistributedDofHandler`.

Figure 4.10 illustrates how the algorithm described in this section can be used to set up the interprocess communication patterns for determining the correct DOF numbering in first-level ghost elements. It considers a simple two-dimensional mesh comprising 5×5 quadrilateral elements with a polynomial order $p = 1$ that is partitioned among 3 MPI tasks as shown in Figure 4.10a. The local mesh on the MPI task with rank = 2 is shown in Figure 4.10b. This figure shows the owned and ghost elements as well as the owned, remote and invalid DOFs and their respective local DOF indices. Note that the owning processes of the ghost elements are also indicated. By looping over its first-level ghost elements, the MPI process with rank = 2 establishes that it has to receive the correct global indices associated with the local DOFs $\{2, 3, 5, 7, 12, 13, 14, 15\}$ from the task with rank = 0 and $\{9, 16, 17, 22, 23\}$ from the process with rank = 1; Analogously, by looping over its owned boundary elements, the MPI tasks identifies the local DOFs that need to be sent to processes 0 and 1. These send vectors are $\{18, 19, 20, 21\}$ for the MPI task with rank = 0 and $\{20, 21, 26, 27, 28, 29\}$ for the process with rank = 1.



(a) Distribution of a 5×5 mesh over 3 MPI tasks.

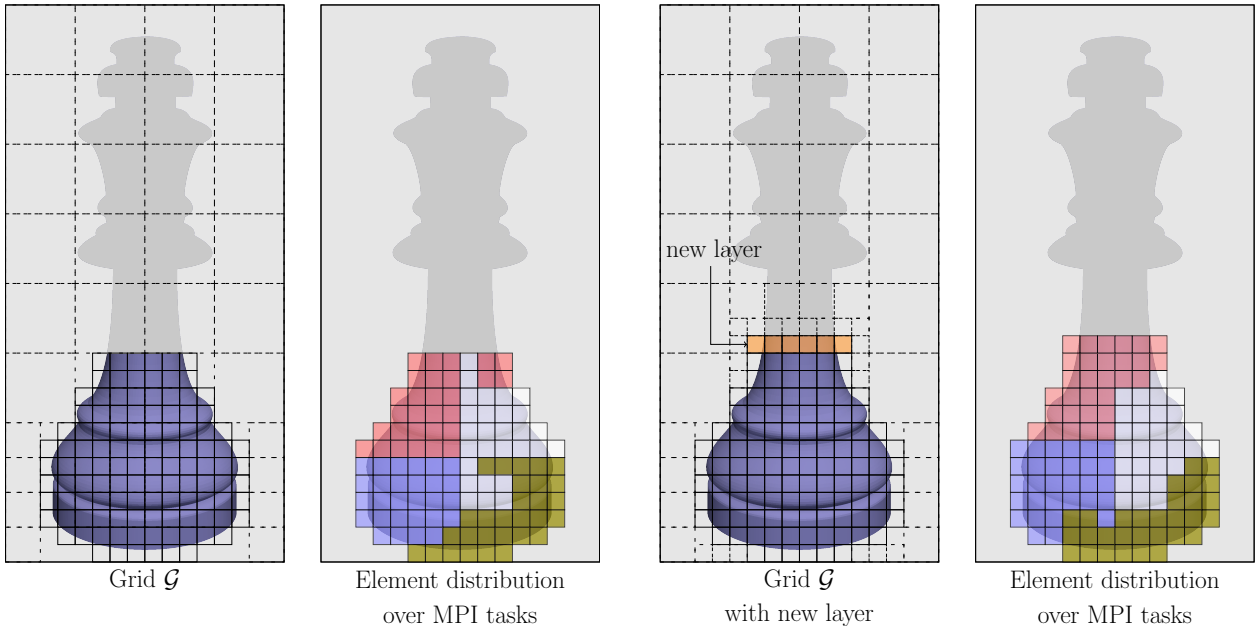
(b) Local mesh $\mathcal{T}_2$ showing the owned and ghost elements as well as the invalid, remote and owned DOFs with their corresponding local indices.

Figure 4.10: An illustration of the creation of the communication patterns in a simple two-dimensional example comprising of a 5×5 mesh partitioned between three MPI processes.

Remark 4.4.3. *The communication patterns set up when enforcing mesh consistency are reused at different junctures in the simulation pipeline such as when updating solution values in ghost elements after solving. Furthermore, it should be noted that the parallel generation of multi-level hp-refined grids does not require the enforcement of a 2:1 mesh balance since the multi-level hp-scheme naturally treats arbitrary levels of hanging nodes.*

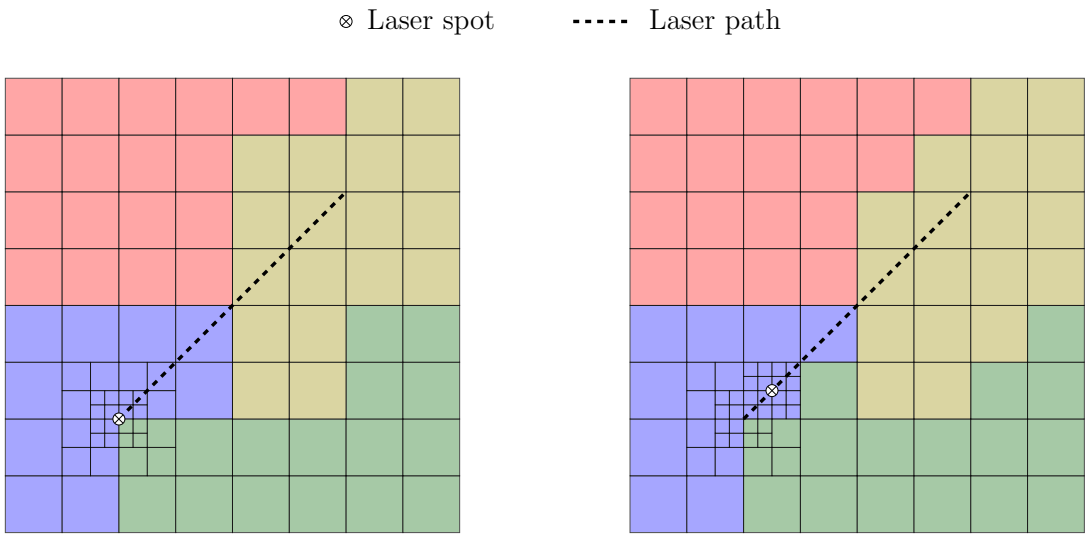
4.4.3 Dealing with dynamic mesh refinement and growing domains

The previous section addressed the creation of a parallel, fully distributed mesh in the context of static finite cell simulations. In this section, parallel algorithms for the management of evolving domains will be presented. These procedures are necessary for time-dependent or dynamic analyses, in which the computational domain is not fixed, but changes over time. Time-dependent domains are common in simulations involving dynamic mesh refinement and in those where the physical domain, Ω_{phys}, grows over time. In such simulations, dynamic redistribution of mesh-related data is often needed to ensure a balanced computational load. This rebalancing process entails the exchange of element data between MPI tasks. The algorithms presented in this section are applied in the simulation of metal additive manufacturing processes, see Section 6.2.



(a) Time step $t = 12$ (b) Time step $t = 13$

Figure 4.11: Illustration of a growing domain in an additive manufacturing process simulation. The union of the active grid cells across all MPI tasks as well as the distribution of elements over four processes is shown for two consecutive time steps. The geometry of the chess piece is obtained from [JP1, 2016].



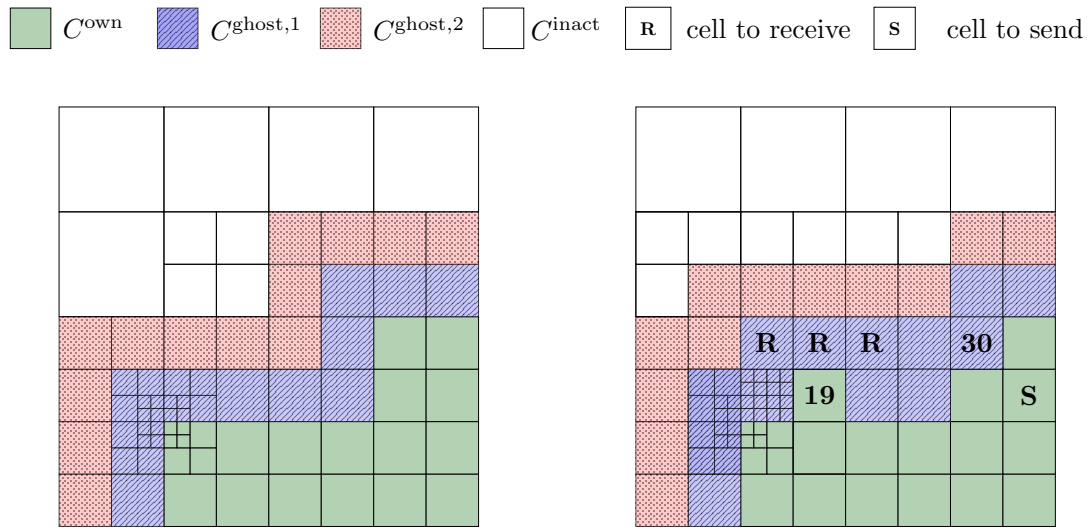
(a) Time step $t = 1$ (b) Time step $t = 2$

Figure 4.12: Illustration of parallel mesh refinement around a laser spot in a parallel simulation of a Selective Laser Melting (SLM) process. The distribution of elements over four processes is shown for two consecutive time steps.

Parallel dynamic mesh management

In this thesis, evolving or transient domains due to domain growth and/or dynamic mesh refinement are considered. Figures 4.11 and 4.12 depict how the distribution of the elements over the MPI tasks can change in these two scenarios. Since the active grid cells and elements residing on an MPI task can change over time, it is necessary to transfer element data, such as the solution and state variables between processors. To this end, a simple algorithm is employed that is applicable for both dynamic refinements and growing domains. This algorithm makes use of the 1-to-1 relations between cells in the grid $\mathcal{G}_{\text{rank}}$ and the mesh $\mathcal{T}_{\text{rank}}$ and first performs operations of the grid before relating them to the mesh elements. It should be noted that the proposed approach is designed for refined scenarios, in which the refinement is guided by geometrical information e.g. the position and path for a laser. The algorithm can, however, be easily modified to allow for refinement based on error indicators and error estimators by adding an extra procedure at the start of each simulation step.

1. At the start of a new simulation step, each MPI process updates the state of the cells in its local grid $\mathcal{G}_{\text{rank}}$. This procedure can involve the activation of formerly inactive cells in the case of domain growth or the refinement and/or coarsening of cells.
2. The previous step generally introduces an imbalance in the computational load across MPI processes. To resolve this, a load balancing step is performed, resulting in a new set of owned grid cells on each MPI process. Although most partitioning tools such as `Zoltan` allow the exchange of element data during (re-)partitioning, this functionality cannot be used in this work. This is because each MPI process not only requires the information of the new elements it receives, but also the information of its first-level ghost elements. For this reason, the proposed algorithm performs this data transfer in a separate step.
3. Next, store the indices and MPI rank of the active cells in the previous simulation step before updating the active cells in $\mathcal{G}_{\text{rank}}$ and creating a new mesh $\mathcal{T}_{\text{rank}}$.
4. Enforce parallel mesh compatibility for all meshes $\mathcal{T}_{\text{rank}}$ following the procedure presented in Section 4.4.2, resulting in a consistent DOF numbering across all MPI processes.
5. The mesh generated in Step 4. contains uninitialized element data, e.g. values of the valid DOFs, or internal problem-dependent state variables that depend on the solution of the previous simulation step. Updating the data of the new elements involves local and global operations. Local updates are performed without communication and entail the transfer of data between elements that existed in both the previous and current mesh. Global updates involve interprocess communication. Before the actual communication stage, each MPI task first determines the element indices that need to be sent to and receive from other MPI tasks. By following the simple rule that element data related to a cell k can only be sent by an MPI process that owned the cell in the previous simulation step, one can setup the interprocess communication patterns. An MPI task's receive operations involve elements not present in the previous simulation step on which valid DOFs are supported. Conversely, send operations involve previously owned elements that were not present on the receiving MPI process and on which valid DOFs are supported. Figure 4.13 illustrates the elements involved in the interprocess communication for the configuration presented in Figure 4.12.
6. Perform non-blocking interprocess communication and update the values in $\mathcal{T}_{\text{rank}}$.



(a) Time step $t = 1$ (b) Time step $t = 2$

Figure 4.13: Cells on the MPI process with rank = 1 that are involved in the interprocessor communication due to the mesh adaptation performed in Figure 4.12. In the second time step, three receive cells adjacent to the cell 19 need to be received by rank = 1, while the cell marked **S** needs to be sent to the processor that owns cell 30.

4.4.4 Numerical examples

The performance of the proposed distributed-memory framework for scalable finite cell simulations is analyzed in the following section. All computations are performed on the SuperMUC-NG system hosted at the Leibniz Supercomputing Center in Garching, Germany. The nodes used have an architecture comprising dual-socket Intel Xeon Scalable Platinum 8168 processors (Skylake). Each node has a total of 48 cores and 96GB of main memory. Furthermore, the following compiler and library versions are used: IntelMPI compiler version 19.0, Trilinos 12.12.1, GNU compiler 7.0, Insight Toolkit 4.12 and Boost version 1.61. The most aggressive level of compiler optimization flags are chosen for the nodal architecture and include -O3 and -funroll-loops among others. The execution time of critical routines in the simulation pipeline is monitored using the cpu_timer implementation in the Boost library [Schling, 2011] while the memory consumption of each MPI task is queried from the operating system. Note that only steady-state examples are considered in this section. The performance of the proposed hybrid framework in transient problems is investigated in Section 6.2.

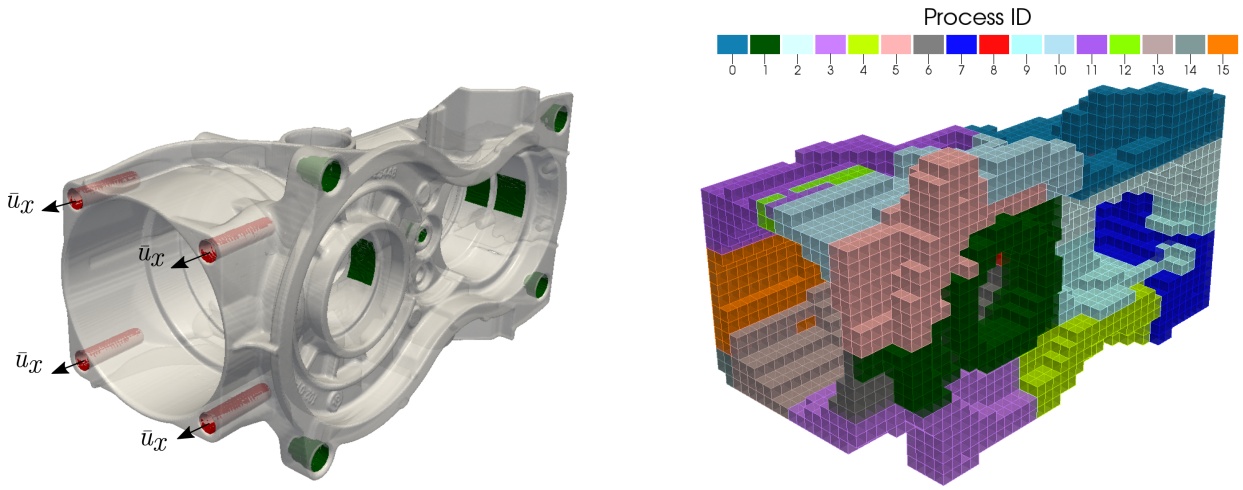
4.4.4.1 Strong scalability: Loading of a gearbox housing

The first numerical example investigates the strong scalability of the parallel algorithms developed in this thesis for large-scale embedded computations. It consists of a linear elastic analysis of an aluminum gearbox housing that was manufactured by die casting. This geometry was first analyzed using the finite cell method in [Monavari, 2011] and the effect of the pores on the stress state is studied in [Jomo et al., 2017] using FCM and the multi-level *hp*-method. Section 5.2.5.3 provides a detailed description of the problem setup and studies the convergence of the arising linear systems when two different preconditioning techniques

are applied. The study at hand focuses on the efficiency of the parallel simulation framework and assesses the scalability of performance-critical routines.

The input data for the example at hand is a CT scan of the gearbox housing with a resolution of $944 \times 592 \times 512$ voxels and a voxel size of $198.7 \times 198.7 \times 198.7 \mu m^3$. Figure 4.14a shows the outer surface of the gearbox that was reconstructed using a marching cubes algorithm [Maple, 2003]. This step is necessary in order to obtain the surfaces required for boundary condition application. The green surfaces in the figure are fully clamped while a horizontal displacement $\bar{u}_x = 0.1$ is applied on the red cylindrical surfaces. Different finite cell meshes can be generated by simply prescribing the number of voxels that should be contained in a single finite cell. Two discretizations are considered in the current study, the first comprising 353 936 quadratic hexahedral elements and 4^3 voxels per element. The second mesh consists of a total of 2 484 111 quadratic hexahedral elements and 2^3 voxels per cell. Both meshes consist of elements from the trunk space and contain approximately 5.2 million and 33.6 million DOFs respectively. Note that the utilized meshes are generated following the procedure described in Section 4.4.1 starting from a coarse grid $\mathcal{G}_0$ with $59 \times 37 \times 32$ cells. Since the underlying geometry of the body is described through a CT scan, the Insight Toolkit [Ibanez et al., 2003] is used to read in the scan file. A voxel-based integration technique using the pre-integration procedure described in [Korshunova et al., 2020] is applied in this example due to its efficiency and suitability for scanned input data. The arising linear systems are solved using the Conjugate Gradient available in the `AztecOO` package of `Trilinos` [Heroux et al., 2005] in conjunction with the elementwise additive Schwarz approach presented in Section 5.2.1.

In the current analysis, the strong scaling of hybrid computations is analyzed. The 48 cores available on each node are partitioned by spanning 8 MPI tasks and a total of 6 OpenMP threads per task. The number of compute nodes is progressively increased from 2 to 64 for the mesh with 5.2 million DOFs and from 16 to 512 for the mesh with 33.6 million DOFs. Figures 4.15 and 4.16 display the execution time for different routines in the simulation pipeline plotted against the number of cores. These routines include the time needed for the generation of the distributed mesh, the enforcement of mesh consistency, the creation of the sparsity pattern, the assembly of the intermediate system matrices, the assembly of the distributed system matrix as well as the preconditioner, and the time spent performing the CG iterations.



(a) Geometry of the gearbox housing (b) Element distribution over 16 MPI tasks

Figure 4.14: Linear elastic analysis of a gearbox housing

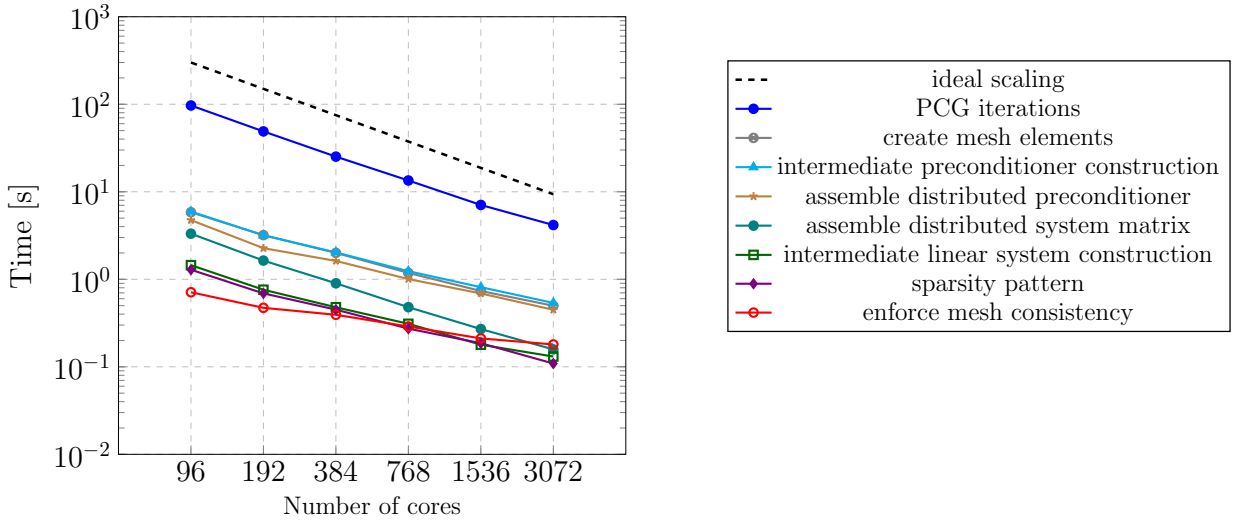


Figure 4.15: Strong scaling analysis of the linear elastic gearbox housing with 5.2 million DOFs.

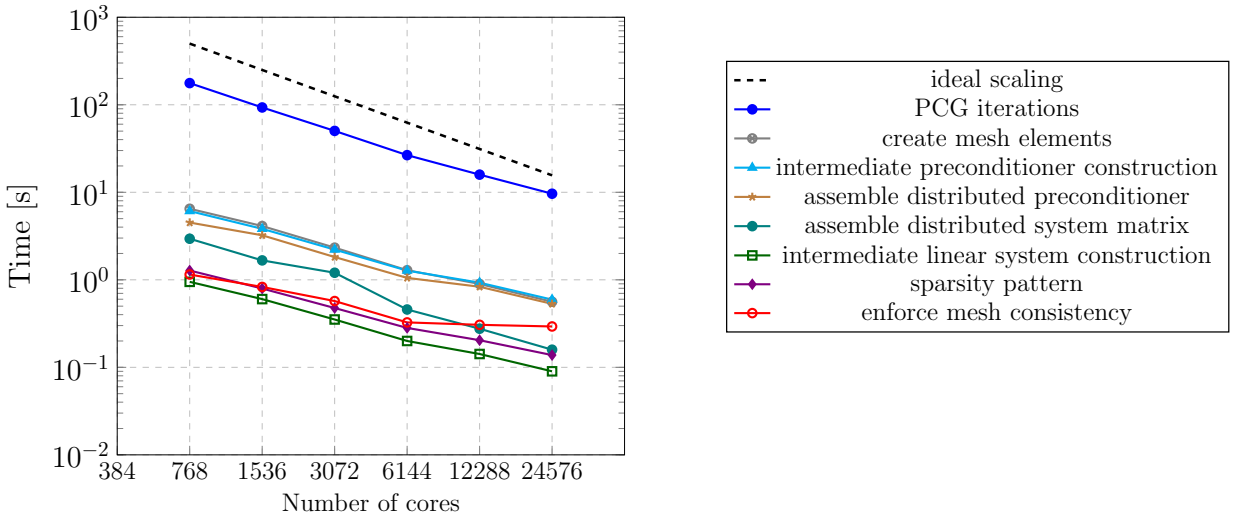


Figure 4.16: Strong scaling analysis of the linear elastic gearbox housing with 33.6 million DOFs.

The solution of the linear system is the most time consuming routine in the simulation pipeline as shown in Figures 4.15 and 4.16. The PCG solver shows good parallel scalability for both discretizations considered in this analysis. Its parallel efficiency lies between 0.99 to 0.73 for the mesh with 5.5 million DOFS and between 0.94 and 0.57 for the mesh with 33.6 million DOFs. The number of iterations performed by the solver is independent of the number of cores and is equals to 1879 for the discretization with 5.5 million DOFs and 3792 for the larger mesh. Other routines that show acceptable scalability include the construction of the sparsity pattern and intermediate linear system. The algorithm for enforcing parallel mesh consistency proposed in Section 4.4.2 also exhibits satisfactory performance requiring a mere 290ms on 24576 cores.

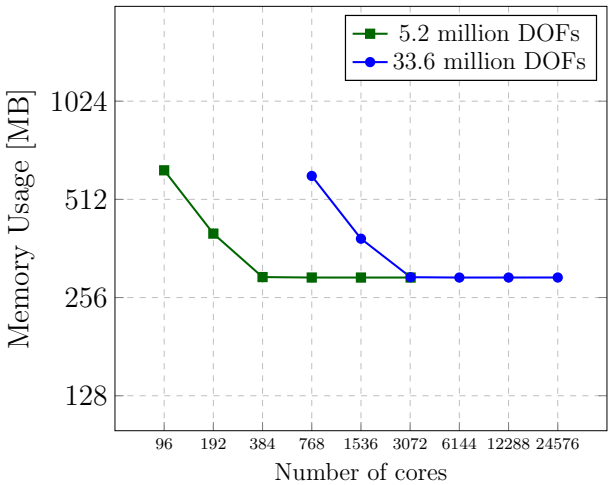


Figure 4.17: Memory usage per core for the analysis of the gearbox housing.

Figure 4.17 shows the highest amount of memory needed by a single core during an entire simulation run. Note that a maximum available memory per core is slightly less than 2048MB.

The memory usage shows satisfactory scaling and levels off at around 294MB. This value corresponds to the amount of memory needed by the Insight Toolkit to process the scan file. The results in Figure 4.17 show that the parallelization strategy proposed in this thesis based on a distributed data structures, has low memory requirements and is suitable for massively parallel machines. This is in contrast to the shared-mesh strategy suggested in [Jomo et al., 2017], that has high memory requirements.

4.4.4.2 Weak scalability: Popcorn benchmark

To assess the weak scalability of the parallel framework presented in this work, a classical benchmark example comprising of a popcorn geometry is considered. This geometry is commonly used in interface problems e.g. [Annavarapu et al., 2012; Chern and Shu, 2007] and immersed analyses e.g. [Burman et al., 2014a]. A level-set function $\phi(x, y, z)$ can be used to describe the surface of the popcorn geometry where

$$\phi(x,y,z) = \sqrt{x^2+y^2+z^2} - r_0 - \sum_{k=0}^{11} A e^{-((x-x_k)^2+(y-y_k)^2+(z-z_k)^2)/\sigma^2}, \tag{4.5}$$

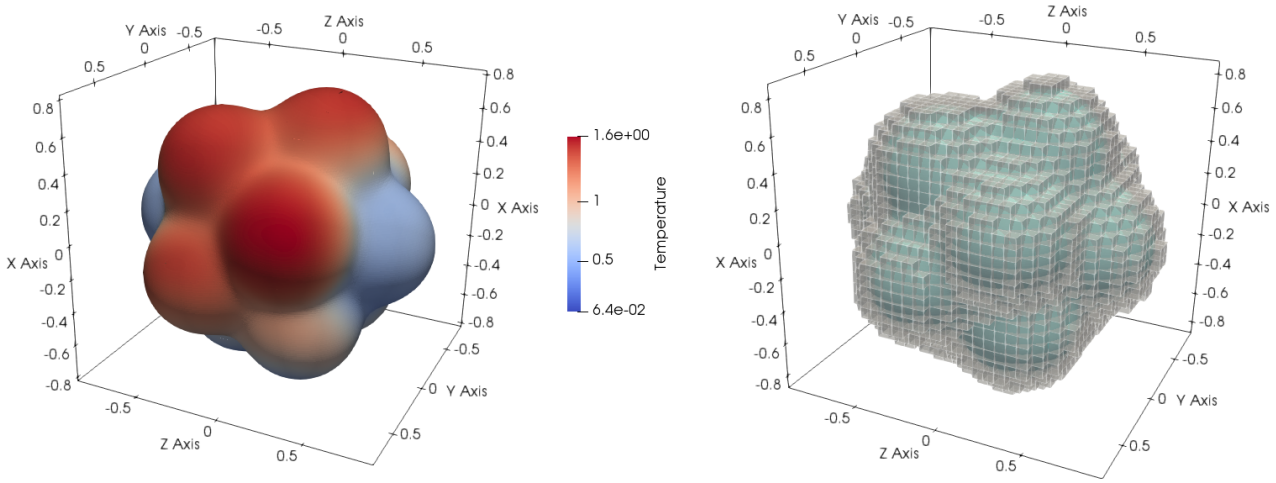
and

$$\begin{aligned}
(x_k, y_k, z_k) &= \frac{r_0}{\sqrt{5}}\Big(2\cos\Big(\frac{2k\pi}{5}\Big), \sin\Big(\frac{2k\pi}{5}\Big), 1\Big), \quad \text{for} \quad k \in [0,4], \\
(x_k, y_k, z_k) &= \frac{r_0}{\sqrt{5}}\Big(2\cos\Big(\frac{(2(k-5)-1)\pi}{5}\Big), \sin\Big(\frac{(2(k-5)-1)\pi}{5}\Big), 1\Big), \quad \text{for} \quad k \in [5,9], \\
(x_{10}, y_{10}, z_{10}) &= (0, 0, r_0), \\
(x_{11}, y_{11}, z_{11}) &= (0, 0, -r_0),
\end{aligned}$$

with the parameters r_0=0.6, A=3 and σ=0.2. The physical domain Ω_{phys} comprises all point lying on the popcorn surface and its interior, i.e. all points with $\phi < 0$. Following [Chern and Shu, 2007], a Poisson problem with a prescribed solution field

$$u(x,y,z) = x^3 + xy^2 + y^3 + z^4 + \sin(3(x^2+y^2)), \quad \boldsymbol{x} \in \Omega_{phys}. \tag{4.6}$$

is considered in the current example. This solution is prescribed through a source term $s = -\kappa^{-1}\Delta u$ and the enforcement of Dirichlet boundary conditions on $\phi = 0$. A marching cubes algorithm is used to obtain the surfaces for boundary condition application. The penalty method with $\beta = 10^4$ is used to apply the Dirichlet boundary conditions and the value of α is set to 10^{-6}. An embedding domain consisting of a cube $(-1, 1)^3$ is used in all numerical investigations.



(a) Popcorn geometry.

(b) Finite cell mesh.

Figure 4.18: Poisson problem posed on a popcorn domain.

A weak scaling analysis is performed to investigate the efficiency of the proposed hierarchical multigrid approach. The simulations are performed on the SuperMUC-NG supercomputer at the Leibniz Supercomputing Center in Garching, Germany. The number of elements is increased in 12 steps while at the same time increasing the number of compute nodes n_{nodes}, from 1 to 2048. The 48 cores within a node are partitioned such that each node has a total of 8 MPI tasks and 6 OpenMP threads per task. The computational domain is generated in a fully parallel manner using the scheme presented in Section 4.4.1 and discretized using hexahedral trunk space elements with a polynomial order $p = 3$. The number of elements per MPI task is approximately 41 000 in all simulations. The simulation with the coarsest mesh is run on 48 cores has a total of 250 336 elements and $1.83 \cdot 10^6$ DOFs while that with the finest mesh is run on 98 304 cores, and has a total of 460 980 224 elements and approximately $3.2 \cdot 10^9$ DOFs. The different linear systems are solved using a parallel CG solver with a p-multigrid preconditioner and elementwise additive Schwarz smoothing.

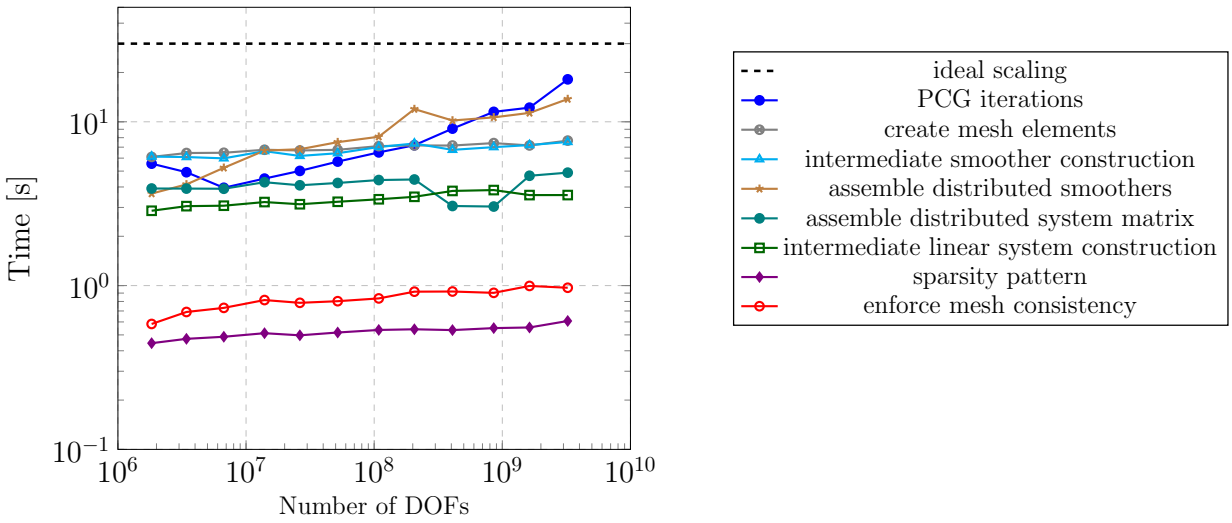


Figure 4.19: Weak scaling analysis of a Poisson problem on a popcorn domain.

Figure 4.19 shows the results of the weak scaling analysis of the Poisson problem posed on the popcorn domain. The maximum time spent in performance-critical routines is plotted against the number of DOFs. The results show that exceptional scaling is achieved for many of the routines in the simulation pipeline, e.g. element creation, integration of the stiffness matrices and the enforcement of mesh consistency. A parallel PCG solver with a p-multigrid preconditioner is used in all simulations, see Section 5.3.2 for details. This solver exhibits satisfactory scalability for the considered linear systems and leads to convergence rates that are independent of the mesh size h, see Figure 4.20. The assembly of the distributed smoothers can be further optimized to improve the overall scalability of the framework.

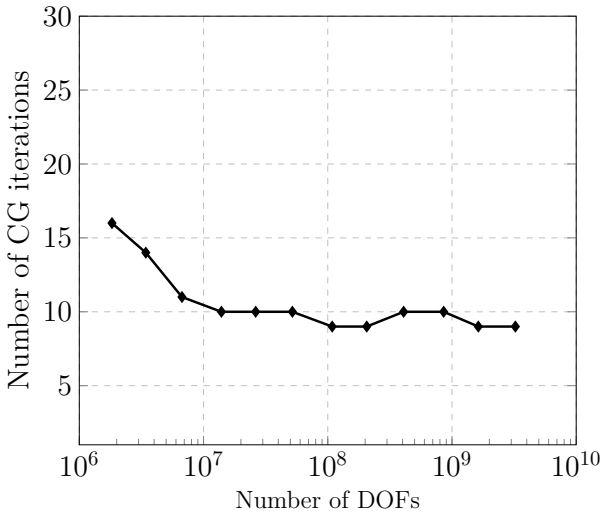


Figure 4.20: Convergence behavior in the weak scaling analysis of a popcorn domain.

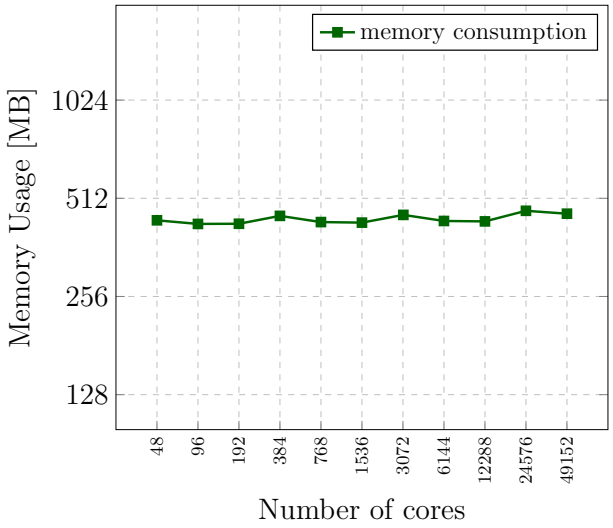


Figure 4.21: Memory usage per core for the Poisson problem on a popcorn domain.

The weak scalability of the distributed data structures is assessed by monitoring the average memory usage per core during the different simulation runs. Figure 4.21 summarizes these results of this analysis. The memory usage per core exhibits good weak scalability as the average memory requirement per core remains around 470 MB per core. The current study shows that the proposed data structures are suitable for massively parallel computations.

Chapter 5

Robust iterative solution techniques for the finite cell method

Immersed finite element systems suffer from ill-conditioning due to the presence of cut cells that lead to the occurrence of small and/or almost linear dependent basis functions. Section 2.3.2 mentions possible remedies to ameliorate these conditioning problems such as basis function manipulation, ghost penalty and preconditioning strategies. The chapter at hand will focus on the development of preconditioning strategies for the finite cell method on uniform meshes and discretizations involving multi-level *hp*-refinement. The preconditioners presented in this work play an integral role in allowing efficient large-scale finite cell computations and are used within the parallelization strategies introduced in the previous chapter. It is well known that the convergence of iterative solvers strongly depends on the conditioning of the system, e.g. [Saad, 2003]. Without tailored preconditioning or stabilization techniques, FCM systems generally show severe ill-conditioning, which practically prohibits the application of iterative solvers. As a result, finite cell computations have for a long time been restricted to the use of direct solvers [Düster et al., 2008; Rank et al., 2012; Ruess et al., 2013, 2014; Schillinger et al., 2012; Schillinger and Ruess, 2014].

5.1 Conditioning analysis of the finite cell method

Although it has been known for many years that small cut elements cause ill-conditioning of FCM systems, it was only until recently that the root cause of ill-conditioning was systematically analyzed and numerically verified in [de Prenter et al., 2017]. In their work, de Prenter et al. show that ill-conditioning is generally caused by basis functions on small cut elements

This chapter is reproduced from [Jomo et al., 2019]: J. Jomo, F. de Prenter, M. Elhaddad, D. D'Angella, C.V. Verhoosel, S. Kollmannsberger, J.S. Kirschke, V. Nübel, E.H. van Brummelen, and E. Rank. Robust and parallel scalable iterative solutions for large-scale finite cell analyses. *Finite Elements in Analysis and Design*, 2019. This paper has been included in the dissertations of the first two authors with the approval of all co-authors and the (co-)promotors. The co-authors and (co-)promotors of the first two authors confirm that the contributions of both John Jomo and Frits de Prenter were essential in this joint publication. The fundamental concepts in the paper have been developed in close collaboration between the first two authors. The specific contributions of Frits de Prenter pertained to the theoretical modification and stabilization of the preconditioner, while John Jomo specifically contributed his expertise in the properties of the *hp*-adaptive basis functions and in the implementation and parallelization.

that can become not only *arbitrarily small* but also *almost linearly dependent*. In the latter case, basis functions that are linearly independent on a full element can become very similar and thus almost linearly dependent when the element is cut. This can arise in scenarios where higher order contributions of basis functions become arbitrary small compared to linear (or lower order) contributions or in cut configurations where dependencies on one parametric dimension (e.g. horizontal or vertical) can become arbitrarily small compared to the other parametric dimensions. These cases lead to a reduction of a higher order or multivariate function to an almost linear or univariate function and by so doing, cause functions to become almost linearly dependent and thereby deteriorate the conditioning of the system. In order to understand the effect of small and almost linearly dependent basis functions in immersed elements, the definition of the condition number given in (2.19) is recalled and shown below.

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\| = \frac{\lambda_{\max}}{\lambda_{\min}} = \frac{\max_{\|\mathbf{v}\|=1} \mathbf{v}^T\mathbf{A}\mathbf{v}}{\min_{\|\mathbf{u}\|=1} \mathbf{u}^T\boldsymbol{A}\mathbf{u}}. \tag{2.19}$$

In FCM, small and almost linearly dependent basis functions generally yield functions in the approximation space that are only supported on small cut elements and have small magnitudes. These functions can become arbitrarily small compared to the magnitude of the vectors that represent them in the isomorphic vector space — which is independent of the volume of the intersection between an element and the physical domain. For this reason, FCM systems generally have $\min_{\|\mathbf{u}\|=1} \mathbf{u}^T\mathbf{A}\mathbf{u} \ll 1$ resulting in a large condition number as per (2.19). The effect of these problematic DOFs, i.e. basis functions that can potentially become linearly dependent, can be systematically analyzed and the aforementioned scaling relation of the smallest volume fraction η and the condition number shown in (2.40) can be derived. This relation holds for second-order problems on uniform tensor product meshes with shape regular elements. Furthermore, de Prenter et al. [2017] show that both Jacobi and Gauss-Seidel preconditioning fail to resolve the ill-conditioning of FCM systems in a robust manner as these methods only target small basis functions but cannot adequately treat almost linearly dependant functions. They present a preconditioner that combines diagonal scaling and orthonormalization of problematic DOFs and show its effectiveness in treating conditioning problems for high-order tensor product uniform meshes. This preconditioner can be viewed as a special type of a class of preconditioners called additive Schwarz preconditioners.

5.2 Additive Schwarz preconditioning for FCM

Ill-conditioning in FCM can be effectively resolved by the use of additive Schwarz preconditioners as shown in [de Prenter et al., 2019a, 2017; Jomo et al., 2019]. This type of preconditioning strategy has been used in different numerical methods and a rich literature exists on the subject in a general setting [Smith et al., 1996; Toselli and Widlund, 2005] or in the context of finite elements, e.g. [Brenner and Scott, 2008; Ferencz and Hughes, 1998].

The main idea behind additive Schwarz preconditioning is the construction of a preconditioner $\mathbf{M}^{-1}$ that is a sparse approximation of $\mathbf{A}^{-1}$ through the inversion and summation of sub-matrices extracted from $\mathbf{A}$. The selection of these sub-matrices is performed by first grouping the basis functions into *groups* or *blocks*. Each block contains indices that correspond to certain basis functions. In general, there is no restriction on the number of groups a single basis function ϕ_k can be present in. With the help of restriction and prolongation operators,

$\mathbf{P}$ and $\mathbf{P}^T$, it is possible to extract sub-matrices from $\mathbf{A}$, invert them and sum the inverse matrices into the preconditioner as summarized by the formula

$$\mathbf{M}^{-1} = \sum_{i=1}^{n_{\text{blocks}}} \mathbf{P}_i \underbrace{\left(\mathbf{P}_i^T \mathbf{A} \mathbf{P}_i\right)^{-1}}_{\mathbf{A}_i^{-1}} \mathbf{P}_i^T \,, \tag{5.1}$$

where n_{blocks} denotes the number of blocks, i the index corresponding to the i^{th} block containing m basis functions such that $\mathbf{A}_i \in \mathbb{R}^{m \times m}$ and $\mathbf{P}_i \in \mathbb{R}^{n_{\text{DOFs}} \times m}$.

The proper selection of the additive Schwarz blocks is essential for obtaining a robust and effective preconditioner, as different choices yield varying preconditioners with different properties. For example, one can assign blocks such that every function is in a single, separate block. This results in the standard Jacobi preconditioner which is not robust when elements are cut, as demonstrated in Section 5.2.1. On the other extreme, one can theoretically assign only one block containing all functions such that $\mathbf{M}^{-1} = \mathbf{A}^{-1}$. Such a preconditioner is optimal in terms of spectral properties, but is prohibitively expensive as it involves inverting the full system. For FCM, it is useful to assign blocks based on the additive Schwarz lemma [Lions, 1988; Matsokin and Nepomnyaschikh, 1985; Smith et al., 1996; Toselli and Widlund, 2005]:

$$\mathbf{v}^T \mathbf{M} \mathbf{v} = \min_{\mathbf{v} = \sum_{j=1}^{n_{\text{blocks}}} \mathbf{P}_j \mathbf{v}_j} \sum_{i=1}^{n_{\text{blocks}}} \mathbf{v}_i^T \mathbf{A}_i \mathbf{v}_i \,. \tag{5.2}$$

In this formulation, $\mathbf{v}_i$ denotes a vector whose length corresponds to the size of the i^{th} block. $\mathbf{P}_i \mathbf{v}_i$ is a prolongated vector whose length corresponds to the size of the full system and that only has nonzero entries at the indices of block i. When each index is in at least one block, for every vector $\mathbf{v}$ there exists a set $\{\mathbf{v}_j\}$ such that $\mathbf{v} = \sum_j \mathbf{P}_j \mathbf{v}_j$. With overlapping blocks, different sets $\{\mathbf{v}_j\}$ with this property exist. The lemma states that the inner product of a vector $\mathbf{v}$ with $\mathbf{M}$ is equal to the minimum of the sum of inner products over all sets $\{\mathbf{v}_j\}$ that sum up to $\mathbf{v}$.

The additive Schwarz lemma can be used to guide the block selection process in FCM. To this end, it is important to recall the cause of ill-conditioning in FCM — basis functions on small cut elements that are small and/or almost linearly dependent. The lemma establishes that a vector $\mathbf{v}$ that has a small inner product with $\mathbf{A}$, will also have a small inner product with $\mathbf{M}$ provided that functions that are almost linearly dependent are aggregated in one block. An elegant way of performing this clustering is by assigning one block for every (cut) element, consisting of all basis functions that are supported on that element, since basis functions generally need intersecting supports in order to be almost linearly dependent. When the blocks are set up in this manner, it so follows from the additive Schwarz lemma that $\mathbf{M}$ inherits the small eigenvalues of $\mathbf{A}$ caused by small cut elements. The preconditioner can therefore effectively treat the fundamental cause of ill-conditioning of the finite cell method, which has already been demonstrated for uniform discretizations in [de Prenter et al., 2019a].

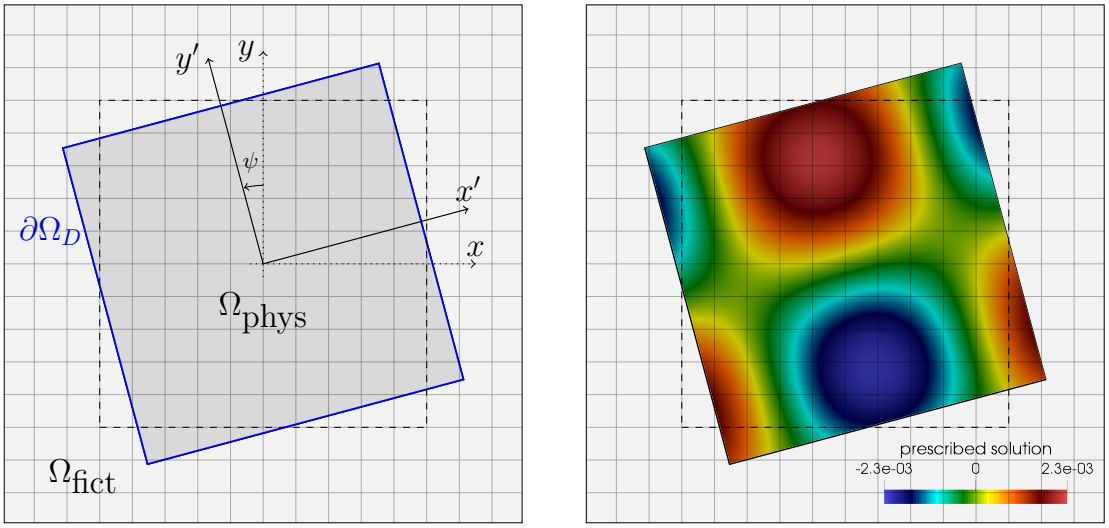
5.2.1 Preconditioning of uniform finite cell meshes

In this section, the effectiveness of additive Schwarz preconditioning is demonstrated for uniform FCM discretizations without refinement by means of a simple two-dimensional example.

The chosen example consists of a Poisson problem posed on a square domain of unit length that is rotated about the origin with respect to a fixed background grid resulting in different cut scenarios as illustrated in Figure 5.1a. The angle of rotation is denoted by ψ. The method of manufactured solutions is applied in this numerical test and a temperature field $\bar{u}$ chosen such that

$$\bar{u} = \frac{1}{2\kappa a^2} \cos(a\, x') \sin(a\, y'), \tag{5.3}$$

with $a = \frac{3}{2}\pi$ and x', y' denoting the coordinates of the rotated coordinate system defined at the center of the domain. A source term s is derived from the manufactured solution following the Poisson equation $-\kappa\Delta u = s$ with $s = \cos(a\, x') \sin(a\, y')$ and applied in Ω_{phys}. Dirichlet boundary conditions are prescribed on all edges of the square domain such that $u = \bar{u}$ on $\partial\Omega_D$. These constraints are enforced using the penalty method with $\beta = 10^4$. The example at hand is suitable for investigating the effect of cut cells on the convergence of iterative solvers. The manufactured solution is chosen such that the total strain energy in Ω_{phys} is independent of ψ.



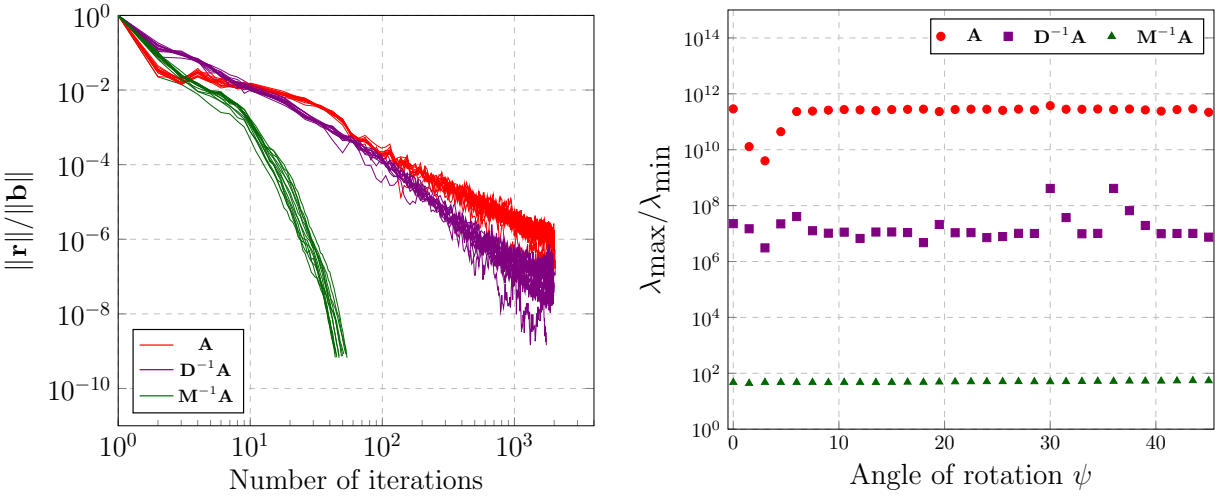
(a) Geometry of the rotating square domain.

(b) Prescribed solution.

Figure 5.1: Rotating Poisson problem with a manufactured solution.

In this study, the angle of rotation ψ is varied in 30 steps from 0° to 45° on a fixed grid of 32×32 quadrilateral elements with a polynomial order of $p = 2$. A conductivity of $\kappa = 10$ is chosen in Ω_{phys} while the value of α in Ω_{fict} is set to 10^{-6}. A Conjugate Gradient solver is used to solve the different systems and the performance of three different preconditioners is investigated: *i)* a scenario where no preconditioner is used, *ii)* a Jacobi preconditioner and *iii)* an additive Schwarz preconditioner constructed in such a way that each element results in an additive Schwarz block. The details of the construction of this preconditioner are summarized in Algorithm 5.1. The results of the current study are presented in Figure 5.2. Figure 5.2a compares the convergence of the relative residual for the different configurations while Figure

5.2b shows the condition number of the preconditioned systems. The simulations with no preconditioner are represented by the label $\mathbf{A}$, while simulations using Jacobi and additive Schwarz preconditioning are denoted by $\mathbf{D}^{-1}\mathbf{A}$ and $\mathbf{M}^{-1}\mathbf{A}$ respectively. A computation is terminated when either a tolerance of 10^{-9} in the relative residual is reached or when the number of iterations is equal to the system size.



(a) Convergence of the relative residual.

(b) Comparision of the condition numbers.

Figure 5.2: Study of the effect of cut cells on the conditioning and convergence behavior of a Conjugate Gradient solver with three different preconditioning techniques.

The study at hand asserts the well-known fact that unpreconditioned FCM systems suffer from severe ill-conditioning as none of the unpreconditioned systems converge to the desired error tolerance. Jacobi preconditioning works well for discretizations that are not prone to almost linear dependencies but performs poorly in scenarios where these phenomena occur. This preconditioning technique can, therefore, be considered not robust with respect to severely cut cells. Additive Schwarz preconditioning adequately deals with the conditioning problems caused by cut cells in this example. It results in a conditioning that is independent of the cut configuration and a convergence behavior that is not significantly influenced by the cut cells.

A detailed analysis of the effectiveness of the different preconditioners can be performed by comparing the spectra of the preconditioned systems. Figure 5.3 shows the spectra of the unpreconditioned system $\lambda(\mathbf{A})$, the diagonally scaled system $\lambda(\mathbf{D}^{-1}\mathbf{A})$ and a system preconditioned with the additive Schwarz preconditioner $\lambda(\mathbf{M}^{-1}\mathbf{A})$ for an angle of rotation $\psi = 45°$. The largest eigenvalue in the unpreconditioned system is in the order of $\beta h \approx 10^4 \cdot 32^{-1} = 312.5$, while the smallest eigenvalue is close to the value of α. Jacobi preconditioning is able to reduce the magnitude of the largest eigenvalue as shown in Figure 5.3. This preconditioner, however, fails to detect basis functions that are almost linear dependent and does not fully treat all the problematic eigenmodes on the cut elements. The additive Schwarz preconditioner effectively acts on both the large and small eigenvalues, yielding a better clustering of the eigenvalues compared to the other two preconditioners. This clustering explains the improved performance of the Conjugate Gradient solver reflected in Figure 5.2a.

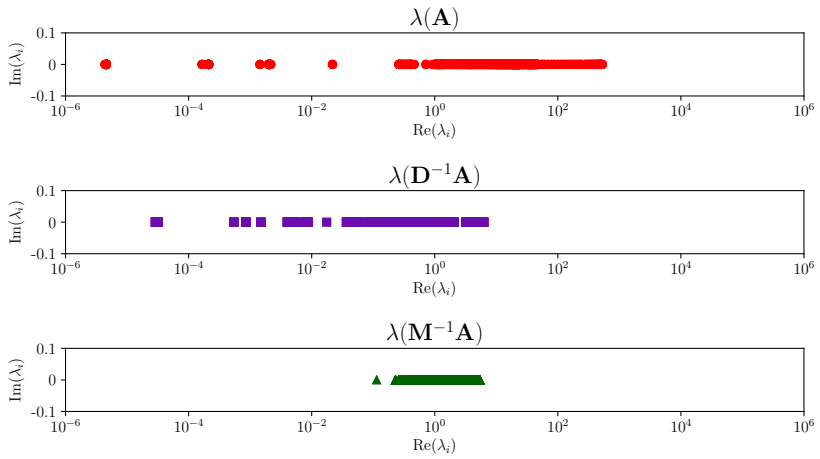


Figure 5.3: Comparison of the eigenvalues of the preconditioned systems in the rotating plate example. The mesh comprises 32×32 elements with a polynomial order $p = 2$ and an angle of rotation $\psi = 45°$.

5.2.2 Preconditioning of multi-level *hp*-refined finite cell meshes

The previous section portrayed how the ill-conditioning of FCM systems on uniform tensor product meshes can be adequately treated using additive Schwarz preconditioning. The utilized preconditioner is constructed by inverting sub-matrices of basis function groups selected in an elementwise manner. This strategy has been successfully applied to problems in linear elasticity [de Prenter et al., 2017; Jomo et al., 2019] and flow problems [de Prenter et al., 2019a].

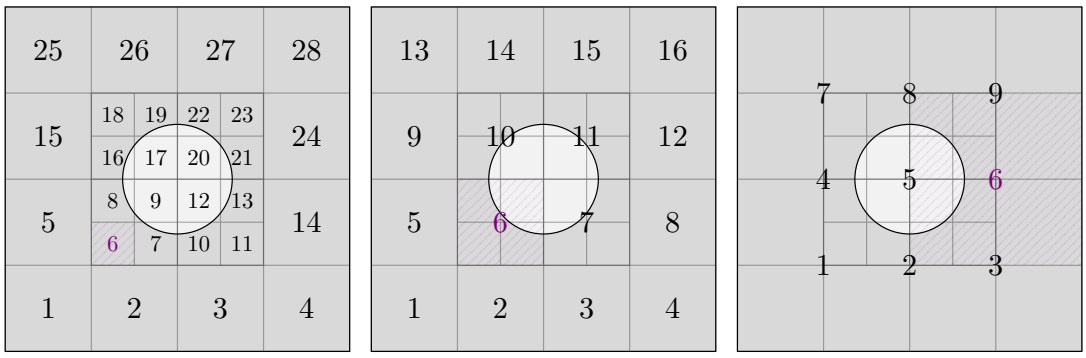
Extending the additive Schwarz preconditioning technique presented in Section 5.2.1 to FCM problems involving multi-level *hp*-refined grids is not a trivial task. To begin with, the presence of elements and basis functions on different levels in the multi-level *hp*-method makes the identification of basis functions that can be linear dependent on cut elements more involving. Considering the requirement that functions that are almost linearly dependent need to be in the same block, multiple possibilities exist for selecting basis function blocks in multi-level *hp*-grids. Moreover, since different refinement patterns are conceivable, it is harder to derive an analytical formula, similar to that for uniform meshes in (2.40), that gives the relation between the smallest volume fraction η, the polynomial order p and the refinement level k that holds in a general setting. Due to the mentioned reasons, a more heuristic approach is adopted in this thesis and several possible preconditioner setups are analyzed experimentally.

Uniform grids and multi-level *hp*-grids have two fundamental differences with regard to the selection of basis function groups. First, on uniform grids the number of basis functions supported on an element is constant, e.g. $(p+1)^d$ for d-dimensional scalar problems using the tensor product space. For multi-level *hp*-grids the total number of basis functions supported on base or leaf elements is, in general, not bounded. This is attributed to the superposed linear hat functions that are present on every refinement level. This superposition results in multiply-defined linear basis functions within an element, as shown in Figure 5.5 . The presence of multiply-defined linear functions does not pose a problem on fully supported elements. In

fact, when the scheme is applied in a boundary conforming manner, these functions actually leads to a better conditioning of multi-level *hp*-systems than classical *hp*-schemes [Zander et al., 2016a]. These functions, however, can lead to severe ill-conditioning in the context of the finite cell method.

The second difference with regard to block selection between uniform grids and multi-level *hp*-grids is that the number of elements that a basis function is supported is bounded in a uniform mesh, i.e. $\leq 2^d$ for C^0-finite elements. With multi-level *hp*-discretizations, it is possible that a (coarse) linear basis function is supported on a large number of (fine) leaf elements, see Figure 3.2. Consider the case that a basis function φ_k is in n different blocks with $n \gg 1$. The unit vector $\mathbf{e}_k$ corresponding to φ_k has a Rayleigh quotient A_{kk} with $\mathbf{A}$. Next we construct a set $\{\mathbf{v}_i\}$ with the value $1/n$ for all vector entries corresponding to φ_k and the value 0 for all other entries. Clearly $\sum_{i=0}^{n_{\text{blocks}}} \mathbf{P}_i \mathbf{v}_i = \mathbf{e}_k$ and from (5.2) it follows that the Rayleigh quotient of $\mathbf{e}_k$ with $\mathbf{M}$ is bounded from above by A_{kk}/n. This shows that a basis function that is in many different blocks yields a small eigenvalue in $\mathbf{M}$, relative to $\mathbf{A}$, and consequently a large eigenvalue in $\mathbf{M}^{-1}$. This reduces the efficiency of the preconditioner, which is demonstrated in Section 5.2.5.3 .

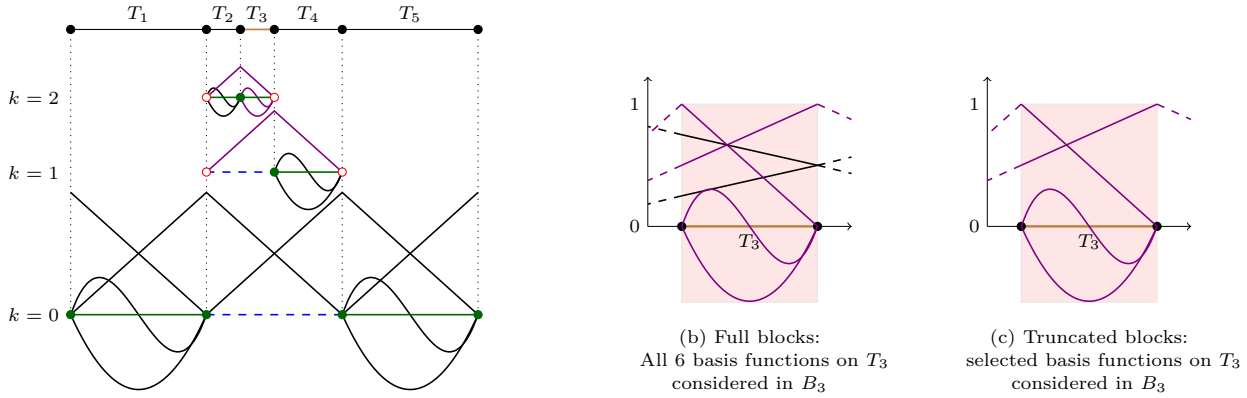
When choosing a preconditioner, a balance is sought between the preconditioner's computational cost, in terms of its setup time, storage requirements and cost of application, and its effectiveness in improving the convergence of an iterative solver. Three possible strategies for the construction of additive Schwarz preconditioners for multi-level *hp*-meshes are investigated in this thesis: *i)* selection of blocks based on leaf elements, *ii)* block selection based on base elements and *iii)* the selection of basis functions based on element patches. Figure 5.4 depicts the three different approaches for setting up the additive Schwarz blocks considered in this thesis. A discussion on the computational cost of setting up $\mathbf{M}^{-1}$ using the different additive Schwarz blocks is given in Section 5.3.3.



(a) Leaf element blocks. (b) Elementwise blocks. (c) Patchwise blocks.

Figure 5.4: Illustration of three different possibilities of choosing additive Schwarz blocks for finite cell grids with multi-level *hp*-refinement. This selection occurs on the granularity of the leaf elements, base elements or using base element patches. The numbers in the figures indicate the number of additive Schwarz blocks and the support of the 6^{th} block is shaded.

Selection of blocks based on leaf elements



(a) One dimensional multi-level *hp*-mesh with 5 leaf elements.

(b) Full blocks: All 6 basis functions on T_3 considered in B_3

(c) Truncated blocks: selected basis functions on T_3 considered in B_3

Figure 5.5: Block selection for multi-level *hp*-grids with the full and truncated blocks of element T_3.

In order to obtain an effective yet efficient additive Schwarz preconditioner from the leaf elements in the multi-level *hp*-method, the basis function blocks have to be carefully selected to ensure that *(i)* the number of blocks a given basis function belongs to is bounded and that *(ii)* the size of each block is small. As already noted in [de Prenter et al., 2019a] it is *sufficient* but not *necessary* to devise a block for every element with *all* basis functions supported on it. In fact, it is enough to overcome the conditioning problems of FCM, due to small eigenvalues on small cut elements, if the blocks contain the basis functions that can potentially become almost linearly dependent. This obviously reduces the size of the blocks, and thereby repairs the large computational cost of setting up, storing and applying the preconditioner. Furthermore, the large linear basis functions that are supported on many leaf elements, and cause large eigenvalues in $\mathbf{M}^{-1}$ as described earlier, are not the basis functions that suffer from almost linear dependencies. Therefore, truncating the blocks to only the basis functions that can become almost linearly dependent when the element is cut resolves both mentioned issues related to the preconditioning of the multi-level *hp*-finite cell method.

To describe the truncation, we first consider a one-dimensional multi-level *hp*-grid with elements of polynomial order $p = 3$. Figure 5.5a illustrates that more than $p + 1$ basis functions can be supported on a leaf element due to multiply-defined linear basis functions. As mentioned previously, these large sets of supported basis functions need to be truncated in order to result in efficient elementwise additive Schwarz blocks. To achieve this, it is first noted that although more than $p + 1$ basis functions can be supported on a single element, each element only has $p + 1$ unique polynomial degrees of freedom. Since eigenvalues with problematically small eigenmodes are only supported on a small cut element, efficient preconditioning only requires the additive Schwarz block of an element to span these $p + 1$ polynomial degrees of freedom. This set of $p+1$ functions must be made up of the $p-1$ higher order basis functions accompanied by two linear basis functions. These linear basis functions depend on the position and refinement depth of a leaf element as illustrated in Figure 5.5a. In the grid shown, two different cases are distinguished: *(i)* the element simply has two linear functions on the

highest level, e.g. T_1 and T_5; *(ii)* the element has one linear function on the highest level and the second linear function is selected by traversing down the element hierarchy and taking the linear function that is linearly independent of the first selected linear function. This can be from a refinement level immediately below the highest level, as is the case for T_3 and T_4, or from an even lower level in the element hierarchy, as is the case for T_2. Figures 5.5b and 5.5c show the block on element T_3 before and after truncation, respectively. Note that when the domain in Figure 5.5a is cut either from the left or the right on any of the elements, the truncated additive Schwarz block described here always spans all functions that are only supported on the cut element and form the problematic eigenmodes for the conditioning of FCM. The truncation in multiple dimensions is achieved by the tensor product of the procedure in one dimension, yielding blocks of $(p+1)^d$ basis functions. Moreover, this choice of blocks not only reduces the size of each block, but also ensures that the number of blocks a single basis function belongs to is at most 2^d. The performance of the truncated and full additive Schwarz blocks obtained from the leaf elements is examined in Section 5.2.5.

Remark 5.2.1. *The size of the preconditioner constructed using truncated additive Schwarz leaf element blocks cannot exceed the size of the original matrix as only basis functions with intersecting supports can be included in the same group. This is in contract to the preconditioner based on full leaf element blocks.*

Selection of blocks based on base elements

Additive Schwarz blocks can also be constructed based on the base elements in a multi-level *hp*-mesh. In this approach, all basis functions supported on a given base element and its subelements constitute an additive Schwarz block. It should be noted that the size of the additive Schwarz blocks selected in this way is not bounded and can become rather large, especially in three-dimensional problems with several refinement levels and high polynomial orders. Although the time needed to set up the preconditioner can be reduced by the use of parallelism, the size and cost of storing the preconditioner may become prohibitively large. In this case, it is possible to apply the preconditioner in a matrix-free manner instead of explicitly constructing $\mathbf{M}^{-1}$. This algorithm can be expressed by considering the action of the preconditioner $\mathbf{M}^{-1}$ on an arbitrary vector $\mathbf{r}$, i.e.

$$\mathbf{z} = \mathbf{M}^{-1}\mathbf{r}. \tag{5.4}$$

Inserting the definition of the additive Schwarz preconditioner from (5.1) in the above function yields,

$$\mathbf{z} = \left(\sum_{i=1}^{n_{\text{blocks}}} \mathbf{P}_i \mathbf{A}_i^{-1} \mathbf{P}_i^T \right) \mathbf{r} = \sum_{i=1}^{n_{\text{blocks}}} \mathbf{P}_i (\mathbf{A}_i^{-1} \mathbf{r}_i) = \sum_{i=1}^{n_{\text{blocks}}} \mathbf{P}_i \mathbf{y}_i. \tag{5.5}$$

Following the above equation, the effect of the preconditioner can be obtained by solving n_{blocks} linear systems and summing up the resulting vectors. These linear systems, $\mathbf{A}_i \mathbf{y}_i = \mathbf{r}_i$, can be solved easily using a direct solver.

Selection of blocks based on element patches

Severe cut scenarios can occur in finite cell meshes with multi-level *hp*-refinement, in which both the leaf element and base element additive Schwarz approaches fail to detect basis func-

tions that can become almost linearly dependent, as shown in Section 5.2.3. Small modes can "slip" through these preconditioners and cause slow convergence. These modes are observed to be not only restricted to a single element but extend to groups of adjacent elements. In order to gain control over these problematic modes that span over adjacent elements, additive Schwarz blocks can be built from element patches. In the proposed approach all elements surrounding a mesh node are considered as a patch as indicated in Figure 5.4c. Numerical examples in Section 5.2.3 show that this preconditioner robustly deals with complex cut scenarios and leads to convergence rates that are independent of the refinement depth. This preconditioner can be constructed explicitly or in a matrix-free manner as explained in the previous section.

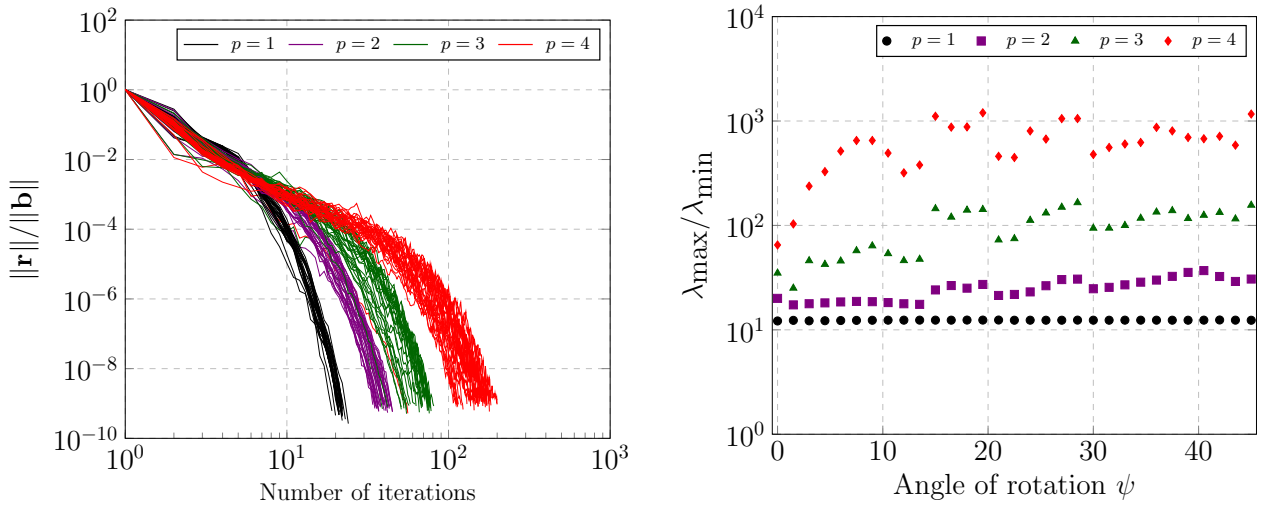
5.2.3 Analysis of the influence of p, h and k on the effectiveness of the additive Schwarz preconditioners

Before elaborating on the implementational aspects of the additive Schwarz preconditioners presented in the previous two sections, the influence of the element polynomial order p, the mesh size h and refinement level k on the convergence of these preconditioners is first analyzed. To this end, the Poisson problem of a revolving square domain considered in Section 5.2.1 is used in a series of numerical examples. A value of $\alpha = 10^{-8}$ is used in all experiments while the value of the penalty parameter is chosen proportional to $h^{-(2p+1)/3}$, following the observations in [Babuška, 1973].

Convergence behavior for different polynomial orders

The influence of the element polynomial order p on the effectiveness of the additive Schwarz preconditioner is assessed for a background mesh comprising 16×16 elements and polynomial orders $p \in [1, 4]$. In analogy to the example in Section 5.2.1, the angle of rotation ψ is varied in 30 steps from 0° to 45°, resulting in different cut scenarios. A penalty parameter $\beta = 2000$ is chosen for the $p = 1$ discretizations and computed in proportion to $h^{-(2p+1)/3}$ for all other polynomial orders.

Figure 5.6 shows the results of the study on the effect of p on the convergence behavior of the additive Schwarz preconditioner. In Figure 5.6a the relative residual is plotted against the number of iterations while the condition numbers of the preconditioned systems are compared in Figure 5.6b. Figure 5.6a shows an increase in the number of iterations for increasing polynomial orders. These results are in accordance with similar studies on immersed elements such as in [de Prenter et al., 2019a]. Furthermore, the results show that the additive Schwarz preconditioner effectively deals with badly cut cells, as there is only a slight variation in the number of iterations for a given polynomial order when ψ is varied. The effectiveness of the preconditioner is also reflected in the condition numbers plotted in Figure 5.6b.



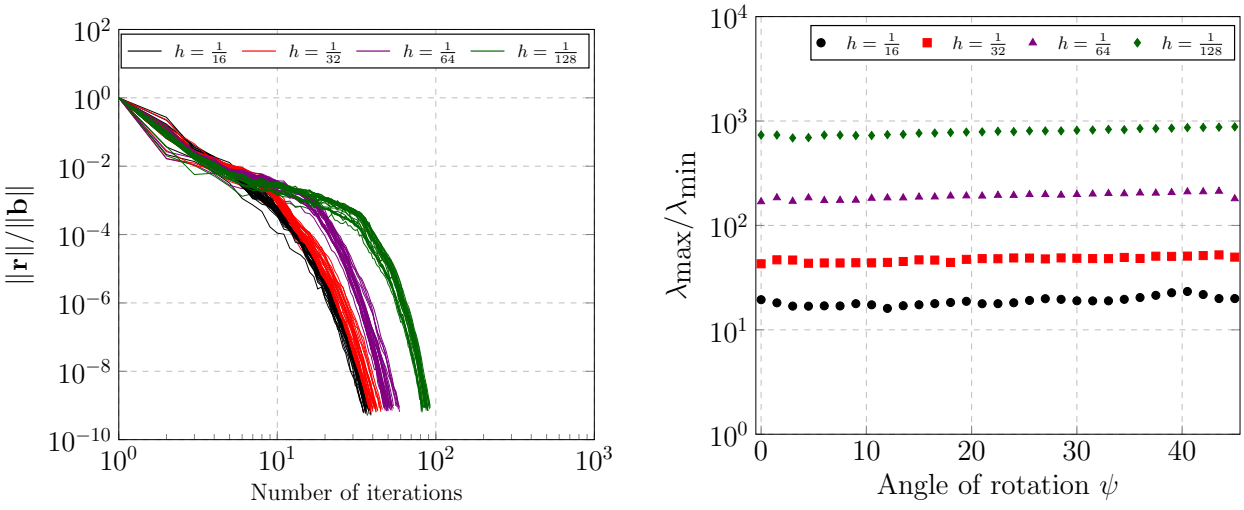
(a) Relative residual.

(b) Condition number.

Figure 5.6: Investigating the influence of the polynomial order on the convergence of the additive Schwarz preconditioner for values $p \in [1, 4]$.

Convergence behavior for different mesh sizes

The influence of the mesh size h on the effectiveness of the additive Schwarz preconditioner is assessed for a fixed polynomial order of $p = 2$ and 30 angles of rotation. The mesh size is varied such that $h = \{\frac{1}{16}, \frac{1}{32}, \frac{1}{64}, \frac{1}{128}\}$. A penalty parameter $\beta = 5000$ is chosen for the coarsest grid with $h = \frac{1}{16}$ and all other values computed in proportion to $h^{-(2p+1)/3}$. The results in Figure 5.7a show an increase in the number of iterations with an increase in the mesh resolution. Similarly, one can see an increase in the condition number with a decrease in the element size in Figure 5.7b. The iteration count for a fixed ψ is proportional to h^{-1} and is accordance with findings in literature [Saad, 2003].



(a) Relative residual.

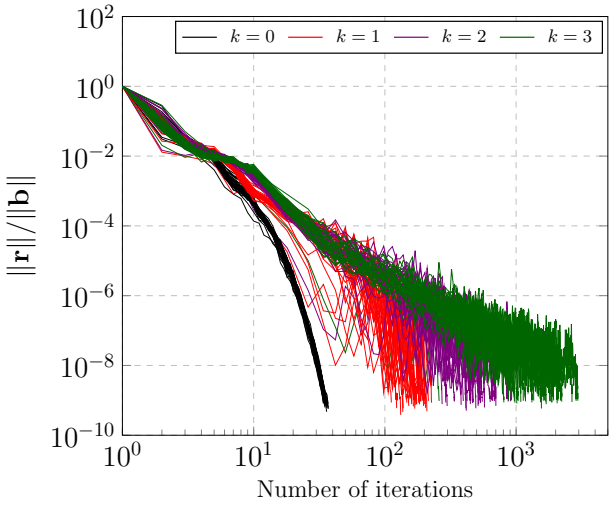
(b) Condition number.

Figure 5.7: Investigating the influence of the mesh size on the convergence of the additive Schwarz preconditioner for values $h = \{\frac{1}{16}, \frac{1}{32}, \frac{1}{64}, \frac{1}{128}\}$.

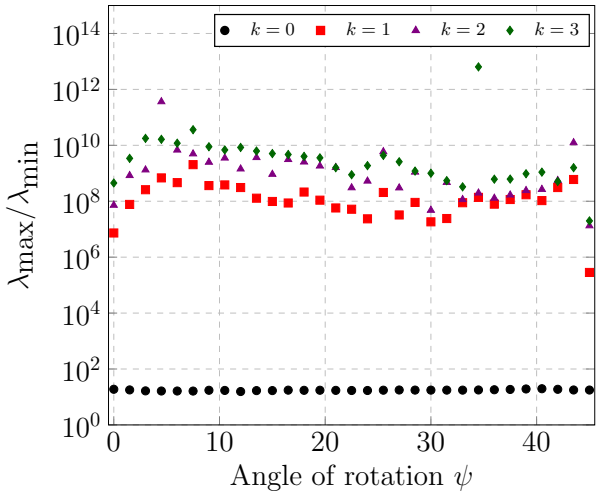
Convergence behavior for different levels of refinement

In the next numerical study, the influence of the refinement depth k on the convergence of a CG solver is compared for the four additive Schwarz preconditioning techniques proposed for FCM systems involving multi-level *hp*-refinement. The preconditioners employed include: the preconditioner with full blocks based on leaf elements; the preconditioner with truncated blocks based on leaf elements; an elementwise preconditioner, where basis functions supported on a base element and all its subelements are taken as one block; and a patchwise preconditioner, which considers all functions supported on a group of elements surrounding a node as a block. The embedding mesh in this study comprises 16×16 elements with a polynomial order of $p = 2$. This mesh is refined recursively towards the boundary of the domain with $k \in \{0, 1, 2, 3\}$. In analogy to the previous investigations in this section, 30 angles of rotation are considered for each refinement level and the results of this study presented in Figures 5.8 to 5.11.

The results in Figures 5.8 and 5.9 show an increase in the number of iterations when the level of refinement is increased for the additive Schwarz preconditioners based on full and truncated leaf element blocks. Recall that the truncated preconditioner limits the number of basis functions in a single block, resulting in smaller additive Schwarz blocks than the full preconditioner. This truncation reduces the computational cost of constructing the preconditioner and leads to an improved conditioning compared to the immersed systems based on full leaf element blocks. This improved conditioning is reflected in smaller condition numbers and smaller iteration counts of the truncated preconditioner compared to the full preconditioner.

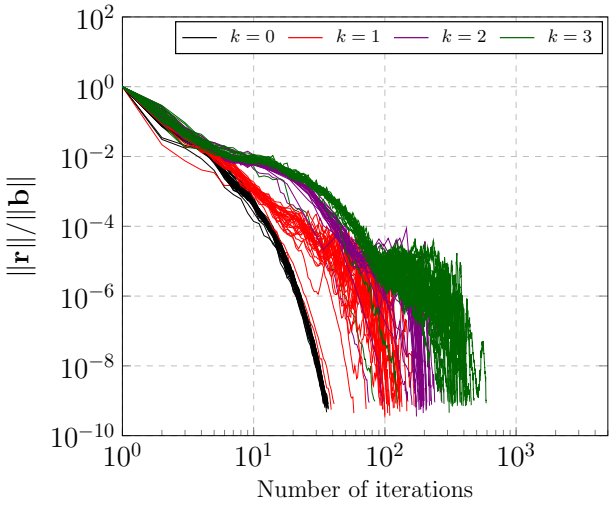


(a) Relative residual.

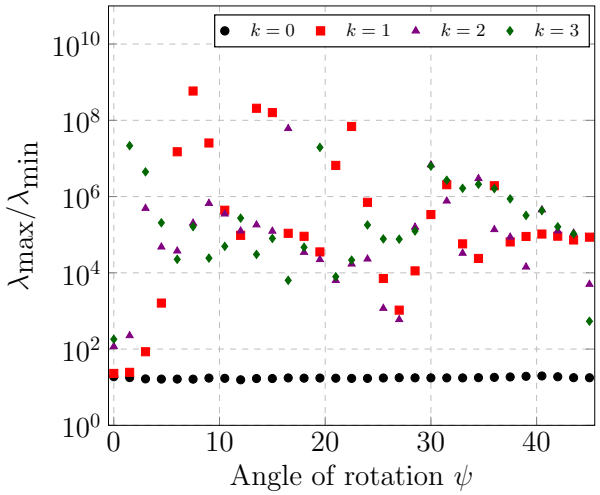


(b) Condition number.

Figure 5.8: Investigating the influence of the refinement depth k on the convergence behavior of the additive Schwarz preconditioner with full blocks based on leaf elements.



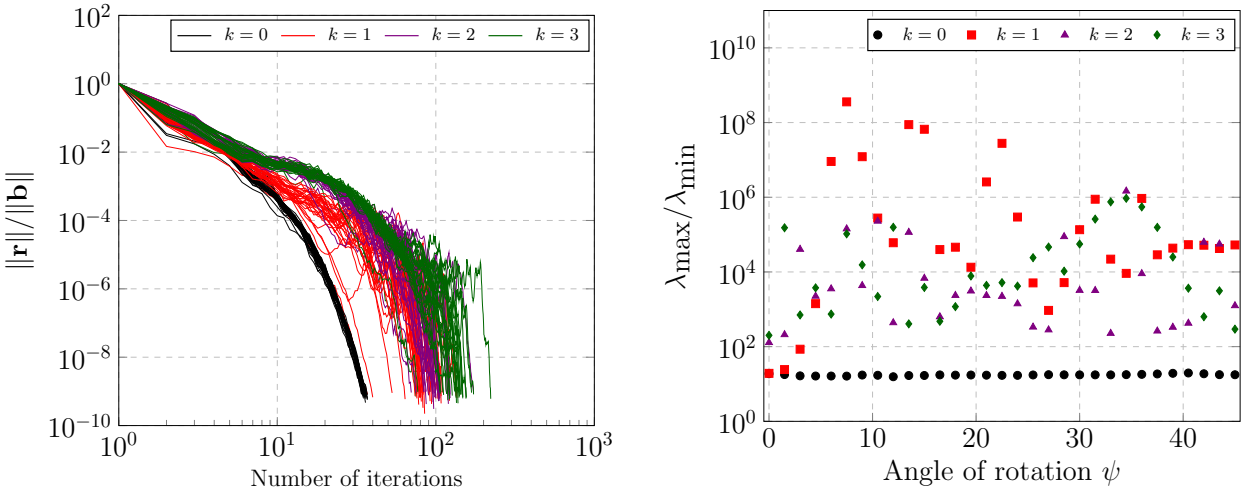
(a) Relative residual.



(b) Condition number.

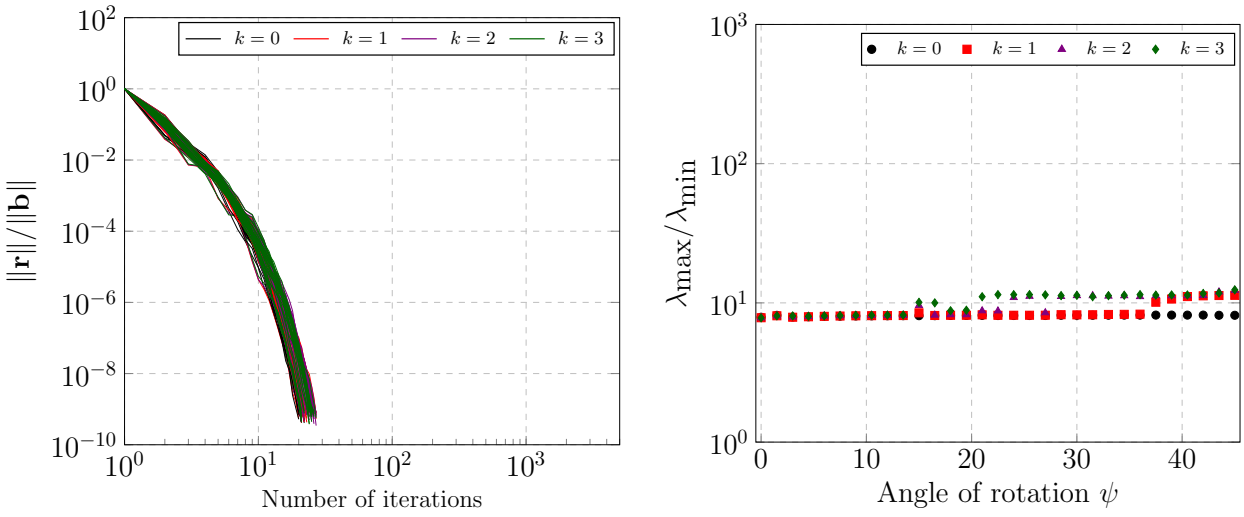
Figure 5.9: Investigating the influence of the refinement depth k on the convergence behavior of the additive Schwarz preconditioner with truncated blocks based on leaf elements.

It is possible to further improve the convergence behavior in the study at hand, by investing more effort in the construction of the preconditioner. As mentioned in Section 5.2.2, this can be achieved by selecting additive Schwarz blocks based on functions supported on a single base element or on a base element patch around a node. Figure 5.10 and 5.11 show the results when preconditioners based on elementwise and patchwise blocks are used.



(a) Relative residual. (b) Condition number.

Figure 5.10: Investigating the convergence behavior of the elementwise additive Schwarz preconditioner for different levels of refinement. Basis functions supported on a base element are considered to be in a group.



(a) Relative residual. (b) Condition number.

Figure 5.11: Investigating the convergence behavior of the patchwise additive Schwarz preconditioner for different levels of refinement.

Figure 5.10 shows that selecting blocks based on base elements leads to smaller condition numbers and fewer iterations than when the preconditioner based on truncated leaf element blocks is used. The preconditioner based on patchwise blocks is the most computationally intensive in its construction, as it involves the inversion of larger basis function blocks than the other preconditioners considered in this study. This preconditioner, however, leads to convergence rates that are independent of the level of refinement as shown in Figure 5.11.

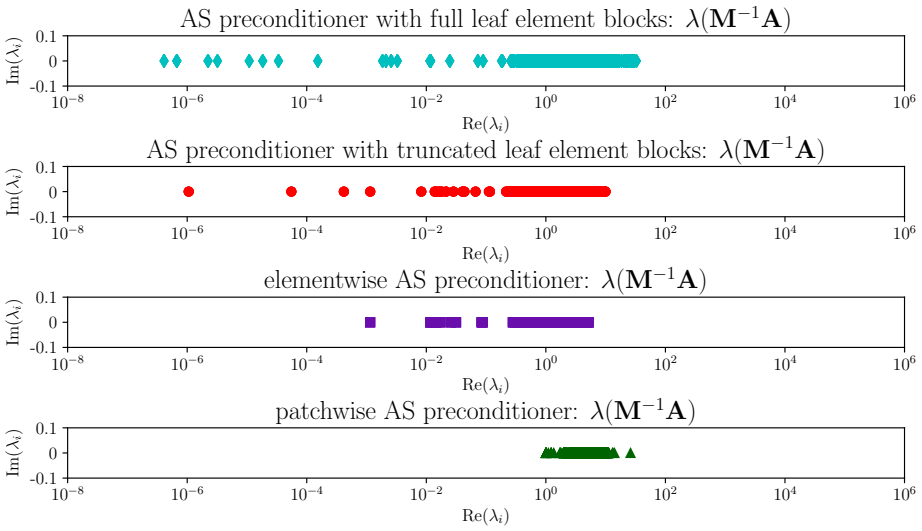


Figure 5.12: Comparison of the eigenvalues of the preconditioned systems for the additive Schwarz preconditioners with blocks based on leaf elements, base elements and element patches. The spectra shown belong to a mesh comprising of 16×16 elements with a polynomial order $p = 2$, an angle of rotation $\psi = 30°$ and a refinement level of $k = 2$.

In order to gain more insight into the convergence behavior of the four preconditioners utilized in this study, a spectral analysis of the preconditioned systems is performed. Figure 5.12 shows an example of the spectra typically obtained for the different preconditioned systems. In general, the preconditioners based on leaf elements and single base elements fail to capture all small modes due to the cut cells. These modes can be said to "slip" through the preconditioner and cause slow convergence. This occurs when using the preconditioners based on full leaf element blocks, truncated leaf element blocks and elementwise blocks. Note that only few modes slip through the truncated and elementwise preconditioners. These modes do not severely affect the convergence of the Conjugate Gradient Method and only result in a few additional iterations. The patchwise preconditioner is able to detect and effectively deal with all problematic small modes on cut elements. Its robustness suggests that problematic modes in immersed multi-level *hp*-discretizations not only occur on single cut elements, but can span over multiple adjacent elements.

Figures 5.13 to 5.16 show a visualization of the smallest and largest eigenmodes of the systems preconditioned with the additive Schwarz preconditioners used in this study. The underlying mesh consists of 16×16 elements with $p = 2$ and $k = 2$. The smallest eigenmode for all preconditioners with exception of the patchwise preconditioner is a function that is only supported on a few cut elements. These small modes fail to be detected by the given preconditioners. In the case of the patchwise preconditioner in Figure 5.16, the smallest eigenmode corresponds to the smallest eigenmode one would expect if standard boundary conforming elements were used. The largest eigenmodes associated with the preconditioners based on full and truncated leaf elements groups are functions that are also supported on cut elements while those associated with the elementwise and patchwise preconditioner are the typical largest eigenfunctions present in boundary-conforming FE approaches.

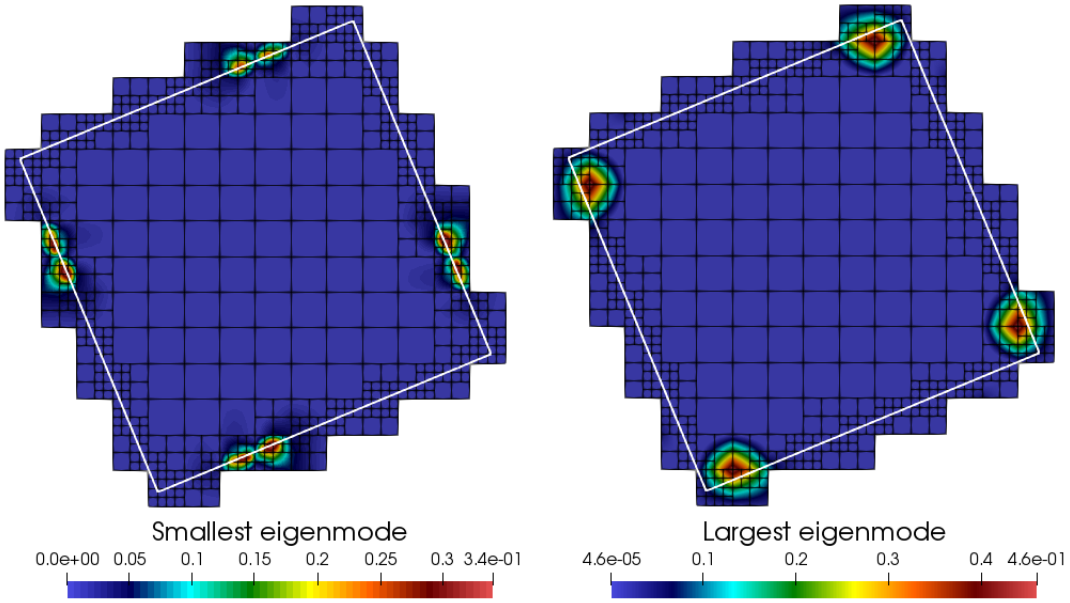


Figure 5.13: Smallest and largest eigenmodes of a finite cell system preconditioned using the additive Schwarz preconditioner based on full leaf element blocks. The smallest and largest eigenmodes are functions supported on badly cut cells.

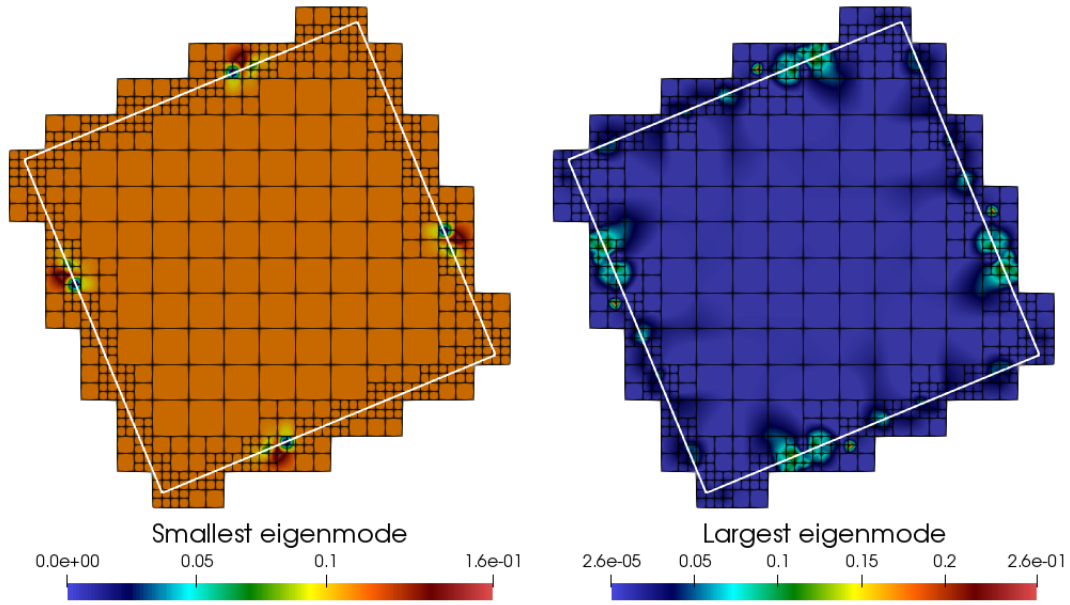


Figure 5.14: Smallest and largest eigenmodes of a finite cell system preconditioned using the additive Schwarz preconditioner based on truncated leaf element blocks. The smallest and largest eigenmodes are functions supported on badly cut cells.

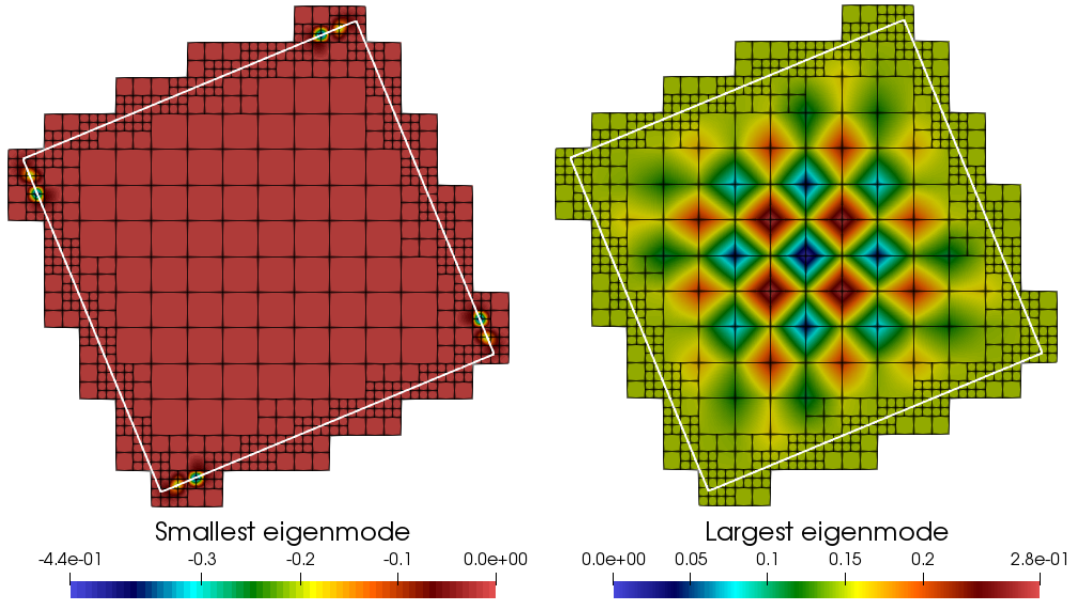


Figure 5.15: Smallest and largest eigenmodes of a finite cell system preconditioned using the elementwise additive Schwarz preconditioner. The underlying mesh has a polynomial order $p = 2$ and is refined in two steps towards the embedded boundary using the multi-level *hp*-method.

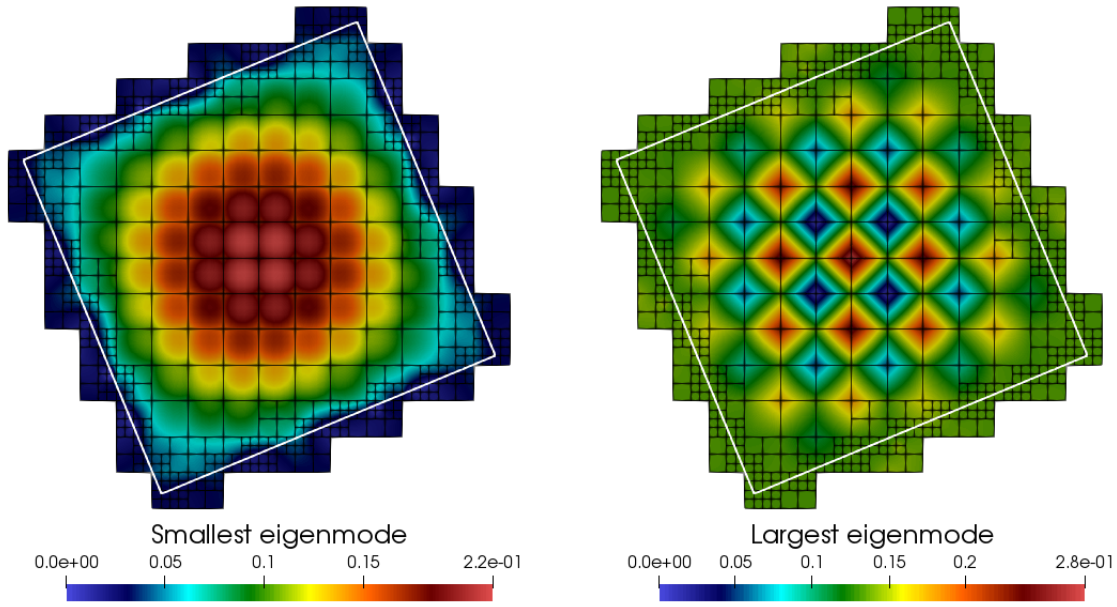


Figure 5.16: Smallest and largest eigenmodes of a finite cell system preconditioned using the patchwise additive Schwarz preconditioner. The underlying mesh has a polynomial order $p = 2$ and is refined in two steps towards the embedded boundary using the multi-level *hp*-method.

Remark 5.2.2. *The study at hand investigates the influence of the polynomial order p, the mesh size h and the refinement depth k, on the performance of the additive Schwarz preconditioners considered in this thesis. Other factors that affect the conditioning and therefore convergence behavior of the immersed systems such as the value of the penalty parameter β, the fictitious domain stiffness α and different methods for strong boundary condition imposition, e.g. Nitsche's method, are not analyzed. The study, however, suffices in showing that the effort needed to solve finite cell systems increases with an increase in the polynomial order, mesh resolution and refinement depth. Furthermore, when dealing with immersed multi-level hp-systems, different types of additive Schwarz preconditioning techniques with varying computational costs and effectiveness in dealing with cut cells are possible. In this work, the truncated preconditioner is used in the numerical examples in Section 5.2.5 that utilize a Conjugate Gradient solver, since this preconditioner is cheap to construct and shows good performance in examples with moderate use of multi-level hp-refinement. The number of additional iterations needed to deal with modes that slip through this preconditioner is small compared to the overall number of iterations. The patchwise additive Schwarz technique is used in Section 5.3.4 as a smoother for hp-refined grids within a multigrid framework. In this case, the patchwise approach helps keeps the number of iterations small.*

5.2.4 Implementational aspects

The following section elaborates on implementational aspects that are advantageous in improving the construction and/or performance of additive Schwarz preconditioners in practice. It also provides a summary of the algorithms needed to construct the preconditioners.

5.2.4.1 Preconditioning of cut cells under a certain volume fraction

The bulk of the computational effort in the construction of the additive Schwarz preconditioners lies in inverting the sub-matrices in (5.1) and storing the inverses, and is proportional to the number of basis function blocks. When a block is devised for *every* element in the mesh, this may lead to a large computational cost. Repairing the specific ill-conditioning effects of FCM does not require a block for *every* element, as can be explained by the additive Schwarz lemma in (5.2) and is demonstrated in [de Prenter et al., 2019a] by only devising blocks for cut elements. The work in [Jomo et al., 2019] explores whether the computational cost can be further reduced by only devising blocks for cut elements whose volume fraction η is smaller than a threshold $\bar{\eta}$. Note that this generally results in a large number of basis functions that lie completely inside the physical domain and are not contained in any of the blocks. Jacobi preconditioning is applied for these basis functions and this operation can technically be interpreted as devising a separate block for every such function. Choosing a smaller value of $\bar{\eta}$ reduces the number of blocks, which yields a more sparse preconditioner and thereby reduces the computational cost. However, smaller values of $\bar{\eta}$ allow smaller untreated elements and with that large condition numbers in accordance with (2.40). This reduces the effectiveness of the preconditioner and increases the required number of iterations in an iterative solver. The effect of $\bar{\eta}$ on the memory usage, number of iterations and computation time is investigated in Section 5.2.5.2.

5.2.4.2 Stabilization of the preconditioner

Constructing the preconditioner in (5.1) entails the inversion of sub-matrices $\mathbf{A}_i$, which should theoretically be symmetric and positive definite, due to the coercive and symmetric weak form of the PDEs considered. Cut scenarios are however possible, in which severely small cuts lead to sub-matrices that contain arbitrarily small eigenvalues. This in turn, can lead to a large difference between the largest and smallest eigenvalues in a sub-matrix. When this difference is in the order of the machine precision or smaller, small eigenvalues that should theoretically be positive can become negative because of rounding errors attributed to the finite precision. Inverting such a matrix with a negative eigenvalue yields an inverse with an arbitrarily large negative eigenvalue that may cause the Conjugate Gradient solver to diverge when the eigenmode associated with this eigenvalue becomes a dominant part of the error. This problem can be remedied by replacing the inverse of $\mathbf{A}_i$ by a *pseudo* inverse $\mathbf{A}_{i,+}^{-1}$ that eliminates eigenmodes in the sub-matrices, when the ratio between the smallest and largest eigenvalues becomes smaller than a certain threshold.

As a starting point for the construction of the pseudo inverse, we consider the spectral decomposition of a SPD matrix $\mathbf{A}_i$ and its inverse $\mathbf{A}_i^{-1}$ written as

$$\mathbf{A}_i = \sum_{k=1}^{n_i} \lambda_k \mathbf{v}_k \otimes \mathbf{v}_k \quad , \quad \mathbf{A}_i^{-1} = \sum_{k=1}^{n_i} \frac{1}{\lambda_k} \mathbf{v}_k \otimes \mathbf{v}_k \,, \tag{5.6}$$

with n_i the dimension of $\mathbf{A}_i$ and λ_k and $\mathbf{v}_k$ the k-th eigenvalue and eigenvector of $\mathbf{A}_i$, respectively. The pseudo inverse that replaces $\mathbf{A}_i^{-1}$ and stabilizes the preconditioning technique is defined following (5.6) and using a cut-off threshold $\lambda_{tres} = \epsilon \max(\lambda_k)$ such that the smallest eigenvalue that is inverted is sufficiently larger than the machine precision.

$$\mathbf{A}_{i,+}^{-1} = \sum_{k=1}^{n_i} \lambda_{k,+}^{-1} \mathbf{v}_k \otimes \mathbf{v}_k \qquad \text{with} \qquad \lambda_{k,+}^{-1} = \begin{cases} \lambda_k^{-1} & \text{for } \lambda_k > \lambda_{\text{tres}} \,, \\ 0 & \text{for } \lambda_k \leq \lambda_{\text{tres}} \,, \end{cases} \tag{5.7}$$

A typical value for this threshold is $\epsilon = 10^{-13}$. It should be noted that the computation of the pseudo inverse involves a eigenvalue decomposition, which is more computationally intensive than the computation of a standard inverse. This routine can, however, be easily accelerated through the use of parallelization and is not a bottleneck in practical computations. Moreover, this pseudo inverse does not need to be applied to all cells but can be applied only to severely cut cells. The numerical example in Section 5.2.5.2 studies the effect of the pseudo inverse on the convergence of an engineering example involving the loading of a spinal vertebra. Note that alternative stabilization techniques can be used in place of a pseudo inverse such as the recursive elimination of problematic basis functions at group level as elaborated in [de Prenter et al., 2019a].

Remark 5.2.3. *The stabilized preconditioner with pseudo inverses generally contains a nullspace. Therefore iterative solvers based on Krylov subspaces such as the Conjugate Gradient method theoretically do not fully converge to the exact solution, unless* $\mathbf{b}$ *is in the range* $R(\mathbf{A})$*, e.g. [Kaasschieter, 1988; Saad, 2003]. However, even if these modes are a part of the solution, this effect is very small in practice because the eigenvalues of the modes disregarded are at least a factor* ε *smaller than the largest eigenvalues in the system. To quantify this effect:* (i) *the relative residual can still converge to at least* ε *(which is set to be smaller than*

the tolerance of the solver), (ii) *the relative energy norm error still converges to at least* $\sqrt{\varepsilon}$ *and* (iii) *the relative preconditioned residual can theoretically still fully converge to* 0.

5.2.4.3 Use of shared-memory and distributed parallelism

The construction and application of the additive Schwarz preconditioner can be sped up though the use of various levels of parallelism. The following section describes how shared-memory and distributed-memory parallelism can be employed to this end.

Shared-memory parallelism

Different shared memory paradigms can be used to accelerate the computation of $\mathbf{M}^{-1}$ on shared memory systems. We opt for the use of OpenMP pragmas [OpenMP Architecture Review Board, 2008] to speed up the construction by parallelizing the loop over the basis function blocks shown in lines 3 to 7 of Algorithm 5.1 . A short study showing the OpenMP scalability of the computation of $\mathbf{M}^{-1}$ will be discussed in Section 5.2.5.2.

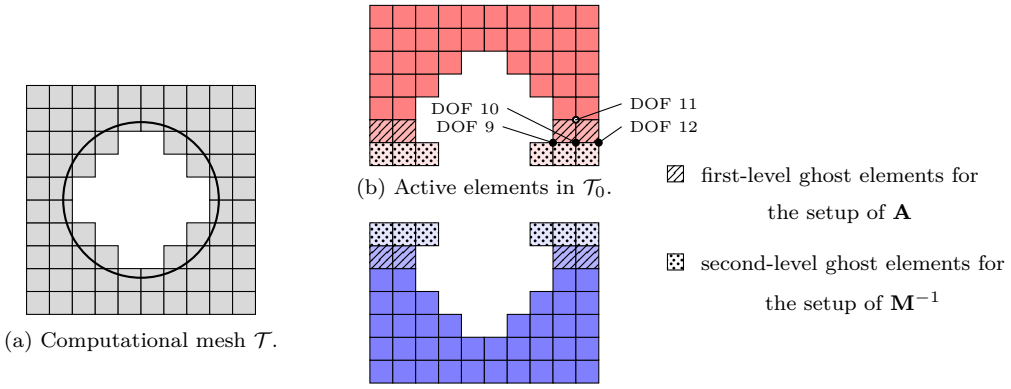
Distributed-memory parallelism

Distributed memory systems are essential for overcoming the time and memory bottlenecks related to the computation of large engineering problems [Ferronato, 2012]. In such computations, the mesh $\mathcal{T}$ is divided over a set of processes that communicate with one another at well-defined synchronization points via message passing libraries such as the Message Passage Interface (MPI) [Clarke et al., 1994]. The locality of the developed preconditioner allows it to be easily used in a distributed memory setting. Chapter 4 explained that each MPI-process is identified by a unique id called *rank* and assigned a portion $\mathcal{T}_{\text{rank}}$ of the mesh, with $\mathcal{T}_{\text{rank}} \subset \mathcal{T}$. The mesh $\mathcal{T}_{\text{rank}}$ consists of elements only residing on the process *rank* and so-called ghost elements which are copies of (boundary) elements owned by the neighboring processes of process *rank*, see Figure 5.17. A hybrid approach combining MPI and OpenMP is adopted, where each MPI-process can use OpenMP threads to speed up local computations on every $\mathcal{T}_{\text{rank}}$.

Although iterative solvers are easier to parallelize than direct solvers, e.g. [Dongarra et al., 1998; Saad, 2003], they face the challenge of robustness w.r.t. the number of processes used [Ferronato, 2012]. A typical caveat of parallel preconditioned iterative solvers is an increase in the number of iterations needed for convergence when the number of processes utilized in a simulation increases. Because of the sensitivity of additive Schwarz preconditioning for the finite cell method caused by the inversion of small cut element contributions, two layers of ghost elements are utilized to ensure that the exact theoretical preconditioner is obtained regardless of the number of processes or the topology of the partitions $\mathcal{T}_{\text{rank}}$. This has the additional advantage that the number of iterations needed for convergence is independent of the number of processes used and thus favors good parallel scalability. It should be noted that the evaluation of the ghost element layers does not significantly increase the setup time of the linear system, since the number of ghost elements on a process is much smaller than the number of active elements. Moreover, since the bulk of the execution time is spent in solving the linear system, the iterative solver's scalability largely governs the overall scalability of the simulation.

Figure 5.17 depicts the partitioning of a mesh between two MPI-processes. The red and blue elements represent active elements on $\mathcal{T}_0$ and $\mathcal{T}_1$ owned by process 0 and process 1 respectively.

The hatched elements constitute the first layer of ghost elements. This layer allows a process to compute the full contributions in $\mathbf{A}$ associated with DOFs it owns without communicating with other processes. This is illustrated in Figure 5.17b where process 0 can independently compute all entries in $\mathbf{A}$ associated with DOF 11 by integrating all the elements around this node. Likewise, the second layer of ghost elements ensures that the exact theoretical preconditioner $\mathbf{M}^{-1}$ can be constructed without communication. The importance of this second ghost-element layer can be explained using Figure 5.17b . Since DOF 11 shares a support with DOFs 9, 10 and 12, process 0 requires the complete entries in $\mathbf{A}$ associated with these DOFs to ensure that all entries in $\mathbf{M}^{-1}$ related to DOF 11 are computed independent of the partitioning as per (5.1). This guarantees that the convergence of an iterative solution method is independent of the number of processes and the element distribution. The above explanation considers uniform meshes, but can trivially be extended to locally refined grids. The parallel performance of a preconditioned parallel Conjugate Gradient solver using the described approach will be investigated in the numerical examples section.



(a) Computational mesh $\mathcal{T}$.

(b) Active elements in $\mathcal{T}_0$.

(c) Active elements in $\mathcal{T}_1$.

Figure 5.17: Partitioning of the mesh in parallel computations for the setup of $\mathbf{A}$ and $\mathbf{M}^{-1}$.

Remark 5.2.4. *The number of options and optimization knobs for tuning the performance of the additive Schwarz preconditioners in order to improve the conditioning of the system, speed up the computation of* $\mathbf{M}^{-1}$ *or reduce its storage requirements may appear to be daunting. It is even possible to apply the additive Schwarz preconditioners implicitly in a matrix-free setting. All these different parameters and settings reflect the complexity of preconditioning hp-refined finite cell systems in a general setting. In practice, when computing an FCM problem for the first time, the following approach is recommended: first start with the simplest variant of the proposed additive Schwarz preconditioner i.e. use a matrix-based approach with elementwise blocks inverted using a standard inverse and* $\eta = 1.0$*. If this works, then the computational time could be possibly improved, if necessary, by measures such as altering the cut threshold as per Section 5.2.4.1. If the solver does not converge, then the elementwise preconditioner that utilizes a pseudo inverse should be utilized. If the solver convergence is still not satisfactory, then the patchwise preconditioner should be employed. Although rigorous tests for different problem types have been conducted to assess the effectiveness of the proposed preconditioners, it should be noted that cut configurations can occur which the preconditioners may not detect. Moreover, It should be noted the cause of slow convergence and solver divergence could lie in*

Algorithm 5.1: computePreconditioner($\mathbf{A}, \{\eta_{K_i}\}, \bar{\eta}$)

```
1  # loop over additive Schwarz blocks
2  for i ∈ n_blocks do
3      # check if the element is badly cut
4      if η_{K_i} < η̄ then
5          A_i = P_i^T A P_i
6          A_{i,+}^{-1} = pseudoInverse(A_i)
7          M^{-1} = M^{-1} + P_i A_{i,+}^{-1} P_i^T
8      end
9  end
10 n = size(A)
11 for j ∈ {1, ..., n} do
12     # check if function not in any block
13     if M_{jj}^{-1} == 0 then
14         M_{jj}^{-1} = 1/A_{jj}
15     end
16 end
17 return M^{-1}
```

other factors such as improper integration or a too coarse mesh.

5.2.4.4 Summary of the preconditioner construction

A flow diagram describing the construction of the additive Schwarz preconditioner $\mathbf{M}^{-1}$ presented in the previous sections is shown in Algorithm 5.1. The volume fraction η_{K_i} of an element is used as a criterion to decide whether a block is devised for the basis functions supported on it. The submatrix associated with this block is then inverted using the "pseudo inverse" presented in Section 5.2.4.2. Note that the algorithm does not explicitly devise a separate block containing one basis function for the basis functions that are not present in any of the blocks but rather performs a separate diagonal scaling step for these basis functions in lines 10 to 16. A major advantage of the proposed preconditioning technique is its suitability for parallel computing, as operations for each block can be performed independently with minimal synchronization. This can be done using both shared and distributed memory parallelism, as described in the Section 5.2.4.3.

5.2.5 Numerical examples

This section demonstrates the suitability of the proposed preconditioner for finite cell analysis of real-life problems. The numerical examples are specifically chosen to highlight various aspects presented in the previous sections. Focus is placed on image-based geometries, an application field where the finite cell method is particularly advantageous in circumventing laborious mesh generation procedures. The first example addresses the extension of the additive

Schwarz preconditioning technique to FCM problems involving multi-level *hp*-refinement by studying a simple example of a cube with a spherical cavity subjected to uniaxial loading. The second example, from the field of biomechanics, considers the loading of a lumbar vertebral body and studies the effect of the pseudo inverse and the volume fraction threshold $\bar{\eta}$ on the computational cost of a preconditioned Conjugate Gradient solver. Finally, the last example brings together FCM, multi-level *hp*-refinement, additive Schwarz preconditioning and parallel computing in the image-based analysis of a die-cast gearbox housing.

The convergence of the PCG solver applied in this section is assessed by monitoring the development of either the relative residual or the relative preconditioned residual defined as

$$\frac{\|\mathbf{M}^{-1}\mathbf{r}_i\|}{\|\mathbf{M}^{-1}\mathbf{b}\|} = \frac{\|\mathbf{M}^{-1}\mathbf{b} - \mathbf{M}^{-1}\mathbf{A}\mathbf{x}_i\|}{\|\mathbf{M}^{-1}\mathbf{b}\|}, \tag{5.8}$$

and when possible the relative error in the energy norm

$$e_i = \frac{\|\mathbf{x}_{\text{ref}} - \mathbf{x}_i\|_{\mathbf{A}}}{\|\mathbf{x}_{\text{ref}}\|_{\mathbf{A}}} = \frac{\sqrt{(\mathbf{x}_{\text{ref}} - \mathbf{x}_i)\,\mathbf{A}\,(\mathbf{x}_{\text{ref}} - \mathbf{x}_i)^T}}{\sqrt{\mathbf{x}_{\text{ref}}\mathbf{A}\mathbf{x}_{\text{ref}}^T}}, \tag{5.9}$$

with a reference solution $\mathbf{x}_{\text{ref}}$ obtained from the parallel direct solver `Intel® Pardiso`, contained in the Intel Math Kernel Library [Intel, 2009]. The subscript i in (5.8) and (5.9) denotes quantities in the i^{th} Conjugate Gradient iteration. Since the systems involved in the following numerical examples are rather large, the effectiveness of the preconditioners used is assessed by monitoring the convergence of the residual and energy error. Computing condition number for the considered systems is prohibitive and is not performed.

5.2.5.1 Compression of a cube with a spherical exclusion

The following example investigates the selection of additive Schwarz block for multi-level *hp*-grids based on the leaf elements. It illustrates the importance of carefully selecting suitable blocks for additive Schwarz preconditioning in FCM simulations with multi-level *hp*-refinement as described in Section 5.2.2. To this end, a simple example comprising of a cube with a spherical cavity under compressive loading is considered. Although this setup yields a relatively small system, with a small number of DOFs ($< 17\,000$), it serves as a good starting point to illustrate the effectiveness of the preconditioning technique developed in this manuscript as it already shows the conditioning problems associated with FCM and multi-level *hp*-refinement.

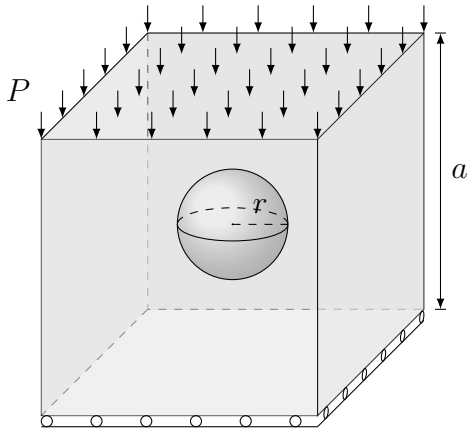


Figure 5.18: Cube with a spherical cavity under compressive loading.

Problem setup

A cube of unit length with a spherical cavity of radius $r = 0.01$ is subjected to a homogeneous pressure load P as shown in Figure 5.18. The cube has a Young's modulus of 70 GPa and Poisson's ratio $\nu = 0.34$. Homogeneous Dirichlet boundary conditions in the normal direction are applied using a penalty parameter $\beta = 10^{10}$. The base mesh consists of $8 \times 8 \times 8$ elements and is refined toward the surface of the exclusion with a refinement depth $k \in \{0, 1, 2, 3, 4\}$. The tensor product space is applied in this example and a polynomial order $p = 3$ is used, resulting in the numbers of unknowns for the different refinement depths given in Table 5.1. Numerical integration is performed using octree partitioning with a tree-depth of 3 relative to the local element size.

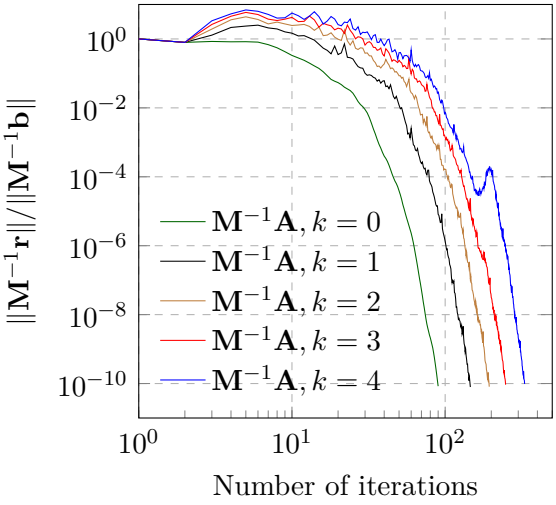
k	0	1	2	3	4
DOFs	13 851	14 541	15 231	15 921	16 611

Table 5.1: Number of degrees of freedom for different refinement depths k.

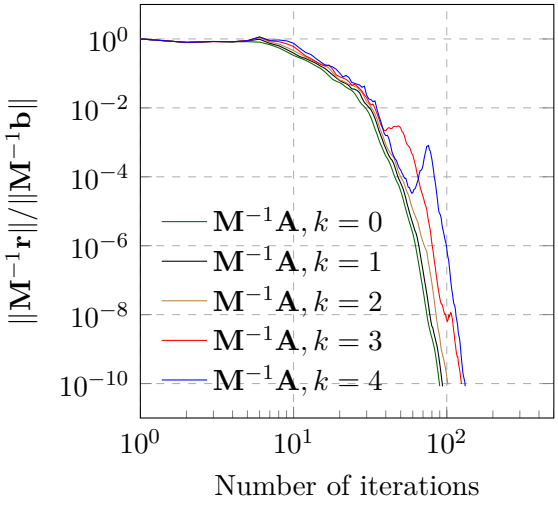
Convergence behavior

The convergence of the PCG solver preconditioned with the full and the truncated blocks is shown in Figure 5.19. The results show that the convergence of the residual closely follows the convergence of the energy. This is generally not the case for ill-conditioned systems such as unpreconditioned FCM, e.g. [de Prenter et al., 2017] or Figure 5.21, and indicates that both preconditioners are robust w.r.t. small cut cells. Figures 5.19a and 5.19c, however, show the effect of local refinements on the convergence speed of the solver when no truncation is used in the construction of $\mathbf{M}^{-1}$. This indicates that the preconditioner with full blocks is sensitive to the refinement depth — i.e. a small increase in the number of unknowns, by approximately a factor of 1.2, leads to a significant increase in the number of iterations needed to reach convergence, which increases by a factor of around 3.6. The intensity of this effect is significantly reduced when the preconditioner with truncated blocks is used as demonstrated in Figures 5.19b and 5.19d. The truncated preconditioner is therefore not only computationally less expensive than the full preconditioner (due to the smaller block sizes), but also yields

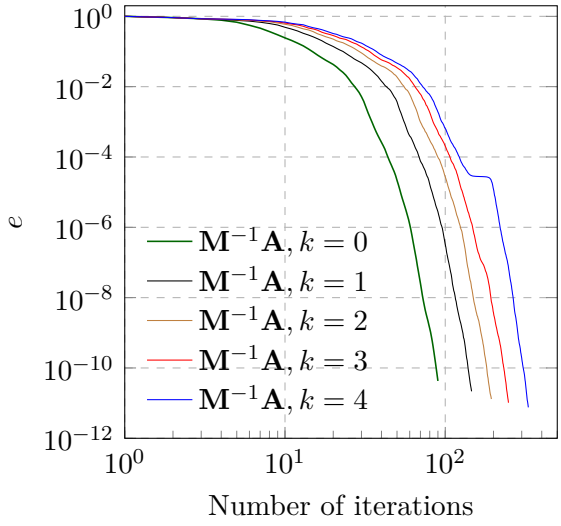
better conditioning. Similar results are obtained using a more complex geometry in the third example in Section 5.2.5.3.



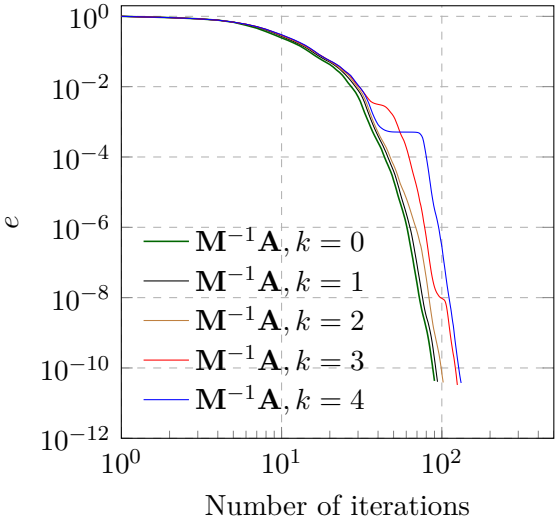
(a) Full preconditioner: residual convergence.



(b) Truncated preconditioner: residual convergence.



(c) Full preconditioner: energy convergence.



(d) Truncated preconditioner: energy convergence.

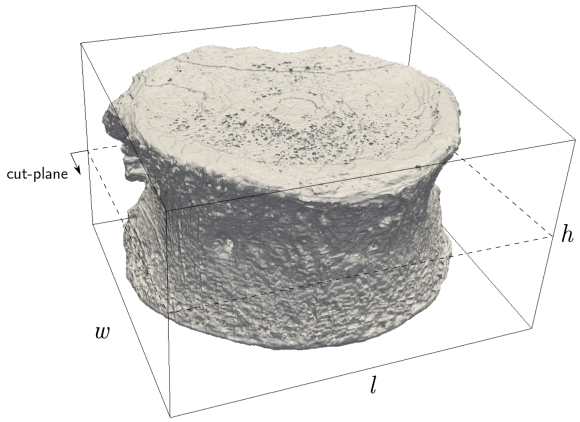
Figure 5.19: Cube with a spherical cavity: Convergence behavior of the PCG solver with full and truncated blocks for multi-level *hp*-refinement. Note that the convergence of the residual is not monotone as the PCG solver minimizes the energy and the residual is only loosely bound to this, e.g. [Saad, 2003] .

5.2.5.2 Image-based simulation of a lumbar vertebra

The applicability of the preconditioner in biomechanical simulations is highlighted in the following image-based simulation of a lumbar vertebra subjected to compressive loading, with the computational setup following [Elhaddad et al., 2018]. Elastostatic computations are

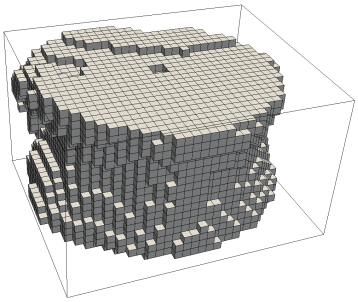
performed with the described setup, utilizing a PCG solver for the solution of the linear system of equations with either no preconditioner, diagonal scaling, or the presented additive Schwarz preconditioner. Various aspects of the iterative solver's convergence behavior are monitored during the simulation for different solver and preconditioner configurations. This example does not consider local mesh refinement. However, the resulting relatively simple problem on a complex geometry is a good test case to examine the stability of the preconditioning technique and the effectiveness of different threshold volume fractions $\bar{\eta}$. Note that on the uniform mesh without local refinements truncation of the blocks is not applicable.

Problem setup

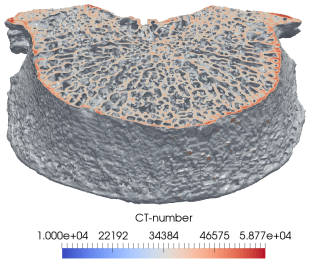


Material	vertebral bone
Young's modulus, E_{bone}	12 GPa
Poisson's ratio, ν	0.3
$l \times w \times h$ [mm]	$47.3 \times 55.3 \times 34.1$

(a) Geometric and material properties of the lumbar vertebra.



(b) Computational mesh of active elements.



(c) Cut showing the porous bone structure and distribution of the CT-number.

Figure 5.20: Setup of the lumbar vertebra example.

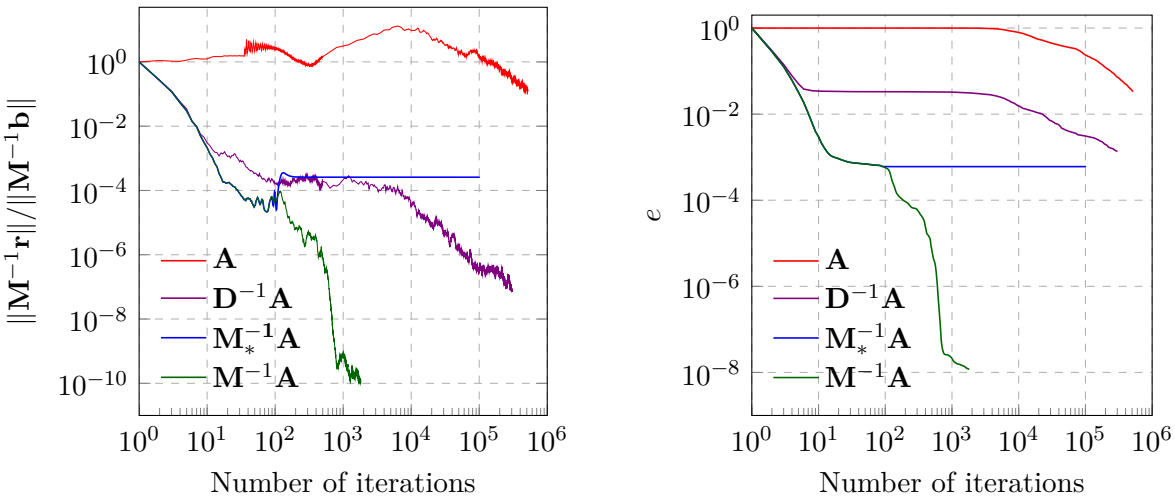
The geometry of the lumbar vertebra is obtained via a high-resolution micro-CT scan of the specimen with a voxel size of $80 \times 80 \times 80$ μm^3. A subsequent segmentation of the scan using `ITK-SNAP` [Yushkevich et al., 2006] yields a model of the vertebral body without the surrounding soft tissue, see Figure 5.20a . It is noted that the segmentation only considers the outer boundary of the vertebra and no distinction is made between cortical and trabecular bone.

A non-boundary conforming discretization is generated using the finite cell method with 10^3 voxels contained in each element resulting in 170 982 elements. Elements completely outside the physical domain are removed in a preprocessing step yielding 75 821 active elements. A polynomial order of $p = 3$ is chosen and the trunk space is applied, resulting in approximately 1.7 million degrees of freedom. Material properties within each element are defined on a voxel-level by applying a threshold on a voxel's CT-number. The CT-number is a measure of the x-ray absorption coefficient that can be related to the density and consequently the stiffness of

the material in a voxel. Material identified as inside the bone structure is assigned a Young's modulus $E_{\text{bone}} = 12$ GPa and Poisson's ratio $\nu = 0.3$, corresponding to values commonly used in literature [Keaveny et al., 2001]. Fictitious material, on the other hand, is assigned the material properties $E_{\text{fict}} = 10^{-11}$ GPa and $\nu = 0.3$ in order to avoid peak stresses in cut elements [Elhaddad et al., 2018]. The surface of the entire vertebral body is recovered using a marching cubes algorithm [Maple, 2003] and thereafter trimmed to provide a triangulation for the superior end-plate, where a uniformly distributed vertical load of 800N is applied, and the inferior end-plate, which is weakly clamped using the penalty method with $\beta = 10^6$. In this example, numerical integration is performed using voxel integration where each element is partitioned into a fine grid of integration sub-cells.

Convergence behavior and stability of $\mathbf{M}^{-1}$

Figure 5.21 shows the values of the relative preconditioned residual and the energy error plotted against the number of PCG iterations performed. The solver is terminated when either a cut-off tolerance in the relative preconditioned residual of 10^{-10} or the maximum number of iterations (300 000) is reached. One can observe that the unpreconditioned system, denoted by the curves labeled $\mathbf{A}$, suffers from ill-conditioning characterized by the slow residual convergence rates and high energy norm errors even after a significant number of iterations. The diagonally preconditioned system, $\mathbf{D}^{-1}\mathbf{A}$, shows an improved convergence behavior but also fails to reach the desired cut-off tolerance within the specified maximum number of iterations.



(a) Relative preconditioned residual.

(b) Error in the energy norm.

Figure 5.21: Convergence of the relative residual and the energy norm errors for different preconditioners in the lumbar vertebra example. The curves $\mathbf{A}, \mathbf{D}^{-1}\mathbf{A}, \mathbf{M}_*^{-1}\mathbf{A}$ and $\mathbf{M}_*^{-1}\mathbf{A}$ represent no preconditioning, Jacobi preconditioning, unstabilized additive Schwarz preconditioning and stabilized additive Schwarz preconditioning, respectively.

The curves $\mathbf{M}_*^{-1}\mathbf{A}$ and $\mathbf{M}^{-1}\mathbf{A}$ depict the systems preconditioned with the preconditioning scheme developed here using the full inverse ($\mathbf{M}_*^{-1}$) and stabilized pseudo inverse ($\mathbf{M}^{-1}$) respectively, with cut-off tolerance $\varepsilon = 10^{-13}$. The importance of filtering out eigenvalues

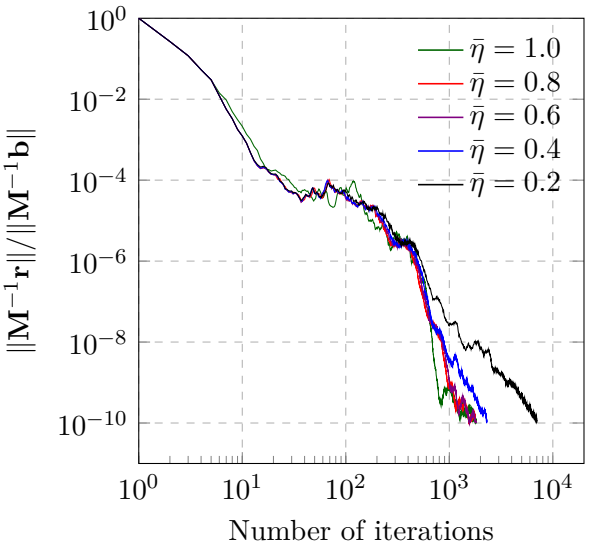
smaller than the machine precision in the local inverses $\mathbf{A}_{i,+}^{-1}$, as described in Section 5.2.4.2 , is clearly visible from these curves in Figures 5.21a and 5.21b . Both $\mathbf{M}_*^{-1}\mathbf{A}$ and $\mathbf{M}^{-1}\mathbf{A}$ exhibit the same convergence until the eigenvectors of $\mathbf{M}_*^{-1}\mathbf{A}$ with negative eigenvalues become the dominant part of the error. It is apparent that the relative energy norm error stagnates and the relative residual diverges in the case of $\mathbf{M}_*^{-1}\mathbf{A}$. These problematic eigenvalues are discarded by $\mathbf{M}^{-1}\mathbf{A}$, resulting in convergence up to the desired tolerance as indicated in the figures. It should be noted that potential loss of accuracy due to the discarding of extremely small eigenmodes is not observed in this example.

It is customary in FCM to pre-process the system and discard all functions that completely lie outside of the physical domain. By stabilizing with the pseudo inverse and setting the fictitious stiffness $\alpha = 0$, this preprocessing step is theoretically not required since the pseudo inverse will ignore these functions. Discarding these functions in a preprocessing step still reduces the computational cost however. It is noted that with α large enough the system does not contain eigenvalues of the order of the machine precision, and the pseudo inverse is equivalent to the full inverse.

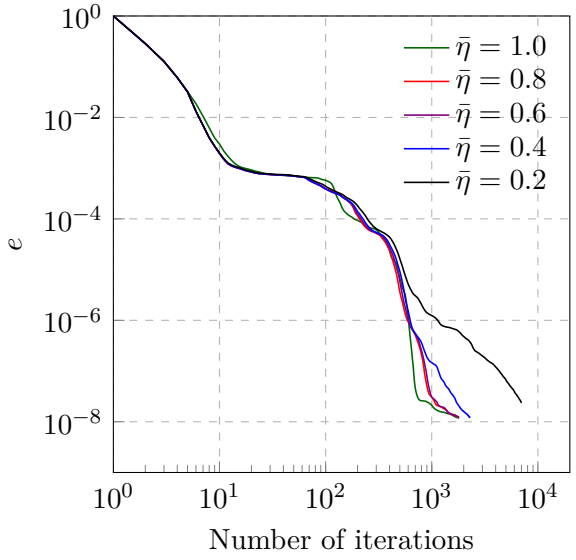
Optimization: Preconditioning only severely cut cells

In [de Prenter et al., 2019a] it is mentioned that the number of blocks used in the setup of the preconditioner can be reduced by considering only elements below a certain volume fraction. This idea is followed in Section in 5.2.4.1 where it is proposed to only invert blocks of basis functions for elements with a volume fraction smaller than a threshold value $\bar{\eta} \in [0.0, 1.0]$. Basis functions that are not present in any of these elements are instead diagonally scaled, to reduce the computational cost. In this section we study the effect of the threshold value $\bar{\eta}$ on the performance of the preconditioner for the lumbar vertebra model described above. To this end, the linear system is solved by a PCG solver for different values of $\bar{\eta}$. A value $\bar{\eta} = 1.0$ corresponds to the preconditioner with a block for every element while $\bar{\eta} = 0.0$ results in a Jacobi preconditioner. It is again stipulated that in this computation the mesh does not contain local refinements, such that the blocks are not truncated.

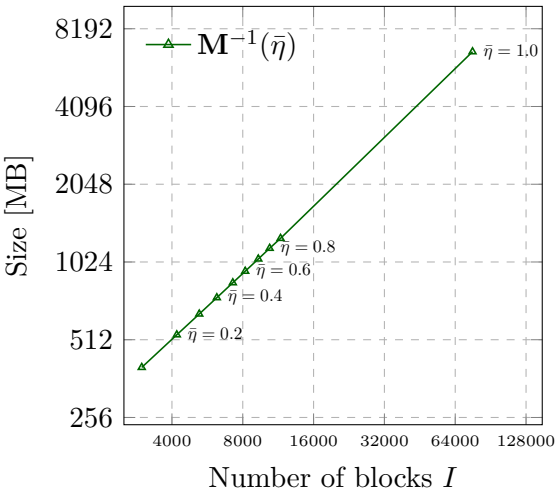
Figure 5.22 depicts the influence of $\bar{\eta}$ on the convergence of the PCG solver, the solution time and memory requirements of setting up the preconditioner. Figures 5.22a - 5.22b confirm that the effectiveness of the preconditioner is maintained as long as problematic basis functions are taken into account when constructing $\mathbf{M}^{-1}$. This holds for $\bar{\eta} \in [0.4, 1.0]$, while values of $\bar{\eta} \leq 0.4$ allow smaller untreated elements and thus reduce the effectiveness of $\mathbf{M}^{-1}$. Reducing the value of $\bar{\eta}$ leads to a sparser preconditioner which requires less storage space as shown in Figure 5.22c , where the size of the preconditioner scales linearly with the number of additive Schwarz blocks. The increased sparsity of $\mathbf{M}^{-1}$ for values of $\bar{\eta} < 1.0$ leads to a reduction in the computational cost of a single PCG iteration, since the matrix-vector multiplication involving $\mathbf{S}$ and $\mathbf{r}$ takes less time. This leads to an overall faster execution time as shown in Figure 5.22d, where the number of iterations needed until convergence is plotted alongside the corresponding solution time for different values of $\bar{\eta}$. The solver converges in less time for $0.6 \leq \bar{\eta} < 1.0$, with the solver converging 1.6 times faster for $\bar{\eta} = 0.7$ than for $\bar{\eta} = 1.0$. This study shows that values of $\bar{\eta} \in [0.6, 0.7]$ yield an effective preconditioner which is sparser and consequently faster than the *complete* preconditioner for $\bar{\eta} = 1.0$ with a block for *every* element. To enable comparison of all presented examples, in the remainder of this chapter the non-optimized choice of $\bar{\eta} = 1.0$ will be considered.



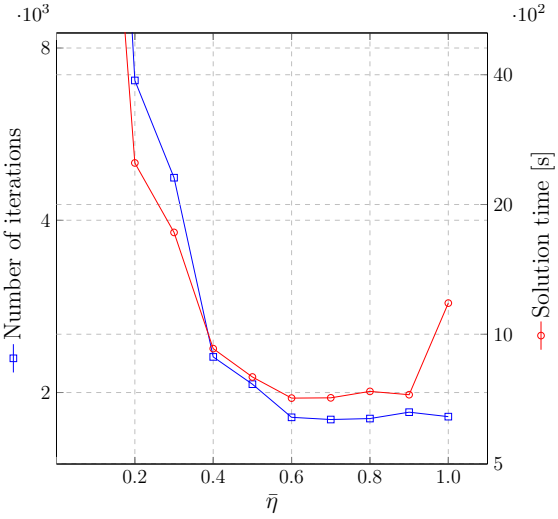
(a) Convergence of the relative residual for various $\bar{\eta}$



(b) Convergence of the energy error for various $\bar{\eta}$



(c) Preconditioner size for different values of $\bar{\eta}$



(d) Convergence behavior for different values of $\bar{\eta}$

Figure 5.22: Preconditioning of cut cells with $\eta_{K_i} \leq \bar{\eta}$.

Computational cost of setting up $\mathbf{M}^{-1}$

Section 5.2.4.3 explains how the construction of $\mathbf{M}^{-1}$ can be accelerated by the use of shared-memory parallelism. This is illustrated in a short study that shows the OpenMP speed up achieved for a variable number of threads involved in the setup of the preconditioner. Three different preconditioners with different numbers of additive Schwarz blocks, and consequently different computational costs, are analyzed. The results of this study are shown in Figure 5.23 . Although good parallel scalability is achieved for the different number of blocks, the parallel efficiency decreases gradually with an increase in the number of threads. This is attributed to the synchronization needed when assembling the entries of $\mathbf{A}_{i,+}^{-1}$ into $\mathbf{M}^{-1}$, (5.1). This effect

can be improved by the use of graph coloring algorithms [Farhat and Crivelli, 1989].

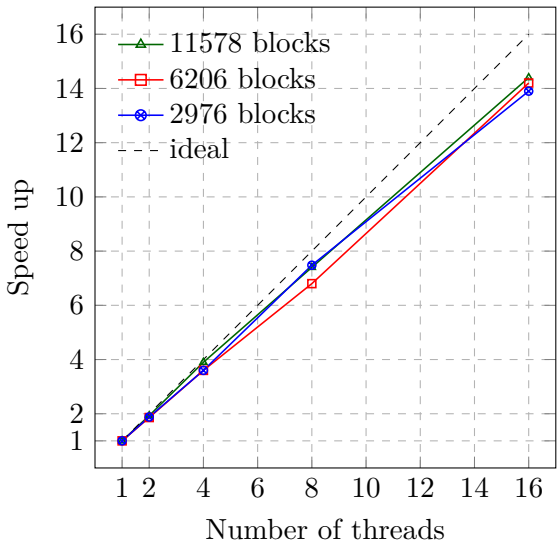


Figure 5.23: Setup of $\mathbf{M}^{-1}$ — OpenMP scalability for different numbers of additive Schwarz blocks.

5.2.5.3 Loading of a die-cast gearbox housing

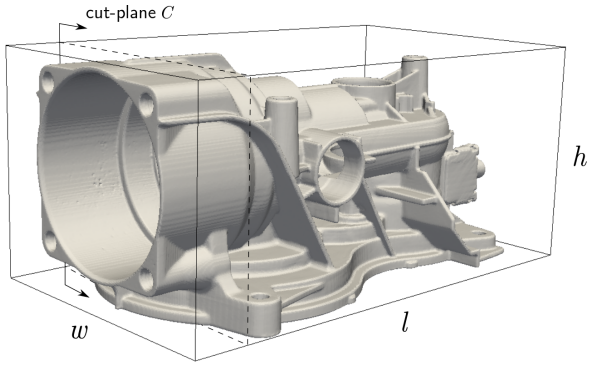
Finally, an industrial example is considered that portrays the suitability of the preconditioner for the solution of large-scale embedded problems involving multi-level *hp*-refinement. The example focuses on the loading of an aluminum gearbox housing produced by die casting, a manufacturing process by which molten metal is forced under high pressure into a mold cavity producing a net-shape structure upon solidification of the molten metal. Die casting is widely used for the production of small- to medium-sized non-ferrous metal parts due to its high production rates, the quality and dimensional consistency of the produced parts and its design freedom, that allows the production of parts with high geometric complexity [Vinarcik, 2002]. The extensive use of die-cast parts is, however, limited by defects such as shrinkage cavities and entrapped gas bubbles that lead to increased porosity in the parts. Pores negatively affect the mechanical properties of die-cast parts and are the preferred sites for fatigue-crack initiation.

Several studies have been conducted on the influence of pore size, position and orientation on the stress state, crack growth and fatigue behavior of aluminum die-cast parts [Ammar et al., 2008; Mayer et al., 2003; Yi et al., 2003]. Finite element analysis of die-cast parts can help quantify the stress concentration in the vicinity of pores through the use of image-based computations Nicoletto et al. [2010]. The limitations of boundary conforming methods in capturing the complex pore morphology can be overcome by the use of the finite cell method as investigated in [Duczek et al., 2015; Monavari, 2011; Würkner et al., 2018].

In this section, we revisit the die-cast example considered in [Monavari, 2011]. The first major modification in contrast to the original example is the use of a preconditioned iterative solver for solving the linear system of equations instead of a direct solver. This modification allows, due to the smaller memory requirement, for computations with a higher resolution

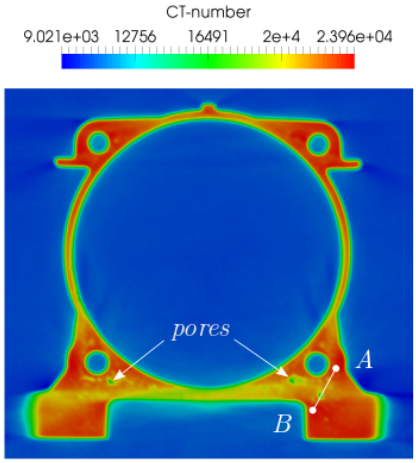
of the die-cast part. Moreover, distributed memory parallelism is exploited to accelerate the performance of the preconditioned iterative solver. The second major modification is the use of multi-level *hp*-refinement to achieve a higher spatial resolution around the pores in the specimen, leading to a better resolution of the complex stress state in these regions.

Problem setup

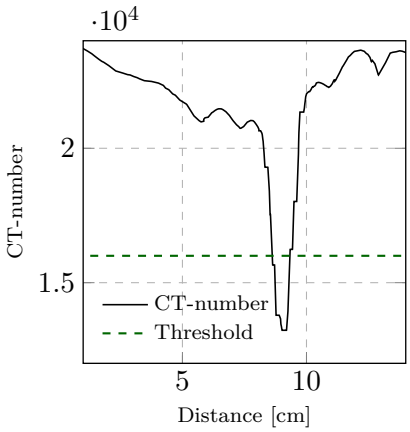


Material	aluminum alloy EN AC-46000
Young's modulus, E	70 GPa
Poisson's ratio, ν	0.29
$l \times w \times h$ [mm]	$175 \times 101 \times 80$

(a) Geometric and material properties of the gearbox housing.



(b) Cut-plane C of the housing CT scan depicting pores in the specimen.

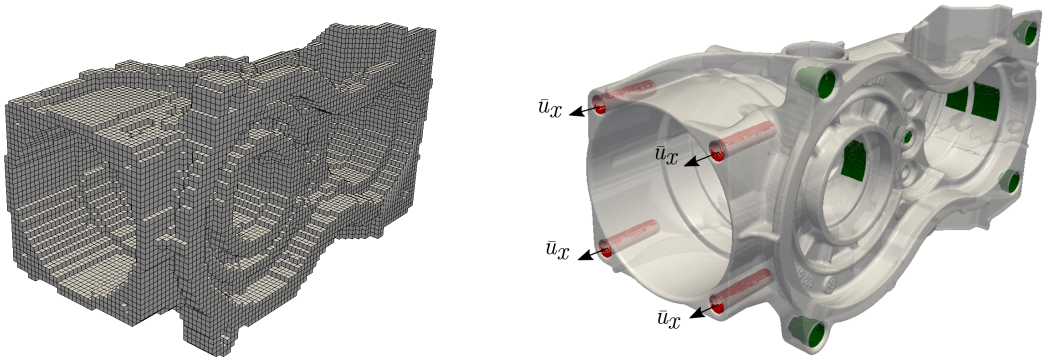


(c) Distribution of the CT-number along the cut-line A-B.

Figure 5.24: Setup of the die-cast gearbox housing example.

The geometry of the gearbox housing is obtained from a CT scan with a voxel size of $198.7 \times 198.7 \times 198.7\ \mu m^3$. A non-boundary conforming mesh is generated with 10^3 voxels grouped to form a single finite cell. Elements completely inside the fictitious domain are eliminated yielding a total of 34 230 active elements in the base mesh before refinement. The classification of the material properties is done by applying a threshold on the intensity values (CT-number) of the CT scan. The threshold is chosen such that the metal material and cavities or internal pores (which are part of the fictitious domain) can be distinguished, as illustrated in Figure

5.24b and Figure 5.24c. Metal material is assigned a Young's modulus $E_{\text{metal}} = 70$ GPa and Poisson's ratio $\nu = 0.3$ while fictitious material is assigned the values $E_{\text{fict}} = 10^{-11}$ GPa and $\nu = 0.3$. Figure 5.25a depicts the computational mesh of the gearbox housing. The discretization is refined towards the surfaces of the interior pores with a refinement depth $k \in \{0, 1, 2, 3\}$. We apply a polynomial order $p = 4$ and the trunk space, resulting in the number of DOFs listed in Table 5.2. A fine grid of integration sub-cells is used for numerical integration as in the previous example. Homogeneous Dirichlet boundary conditions are applied on the dark green surfaces while a displacement $\bar{u}_x = 0.1$cm is applied on the red cylindrical surfaces as depicted in Figure 5.25b. All boundary conditions are enforced weakly using the penalty method with the penalty parameter $\beta = 10^6$.



(a) Discretization of the gearbox housing.

(b) Application of boundary conditions.

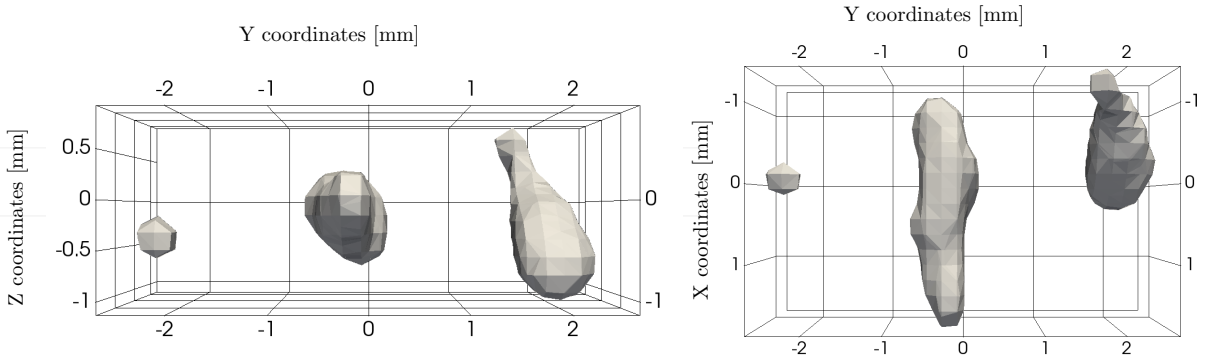
Figure 5.25: Mesh and boundary conditions of the gearbox housing example. All displacements on the dark green surfaces are fixed while a displacement field $\bar{u}_x = 0.1$cm is prescribed on the red surfaces.

k	0	1	2	3
DOFs	1 662 666	1 670 478	1 683 792	1 716 102

Table 5.2: Number of degrees of freedom for different refinement depths k.

Influence of the pores on the stress distribution

Before studying the convergence behavior of the iterative solver for different refinement depths k, we highlight the necessity of local refinements in resolving stress states in the vicinity of small geometric features with an economical number of DOFs. To this end, the three pores indicated in Figure 5.24b are considered. Figure 5.26 shows a smoothed representation of the pore geometries obtained by segmenting the pores gearbox scan using `ITK-SNAP` and exporting the resulting triangulated surfaces.

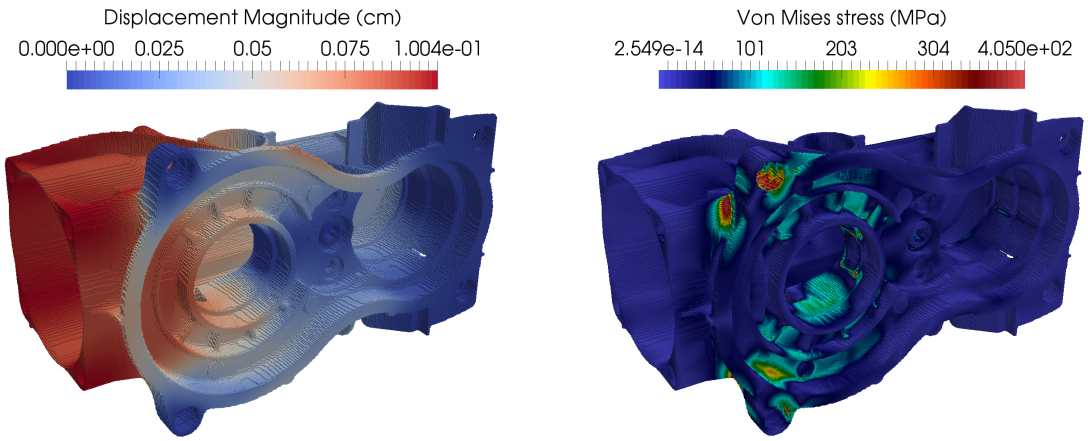


(a) Frontal view of the considered pores.

(b) Aerial view of the considered pores.

Figure 5.26: Geometry of the three pores (left to right) shown in Figure 5.24b.

Different values of the refinement depth k yield comparable results of the overall displacement and stress fields in the gearbox housing with an exemplary result given in Figure 5.27 . Differences arise, however, in the stress fields in the vicinity of the pores as shown in Figures 5.28 - 5.29 . Without local mesh refinement, the chosen mesh is unable to accurately capture the expected stress concentration around the pore boundary, as shown in Figure 5.29a . The use of a globally h-refined mesh would help capture stress concentration around the pores, but is not a viable option since the number of DOFs would increase drastically. Moreover, performing h-refinement would not significantly affect the stress distribution away from the pores, since this state is already adequately represented by the coarse mesh as shown in Figure 5.28 . The stated arguments motivate the use of hp-refinement to capture local solution stress distributions in a computationally efficient manner. The results of the Von Mises stress available in Figure 5.29b and Figure 5.28 show that the location of the peak stress shifts to the boundary of the pores upon application of multi-level hp-refinement, which is physically realistic behavior. Note that this linear elastic example only portrays the importance of refinement in resolving small geometric features and is not an extensive analysis of the stress state around the pores. Such a detailed analysis would require more advanced models e.g. elasto-plastic behavior.



(a) Displacement magnitude. (b) Von Mises stress distribution.

Figure 5.27: Results of the displacement and von Mises stress fields of the gearbox housing.

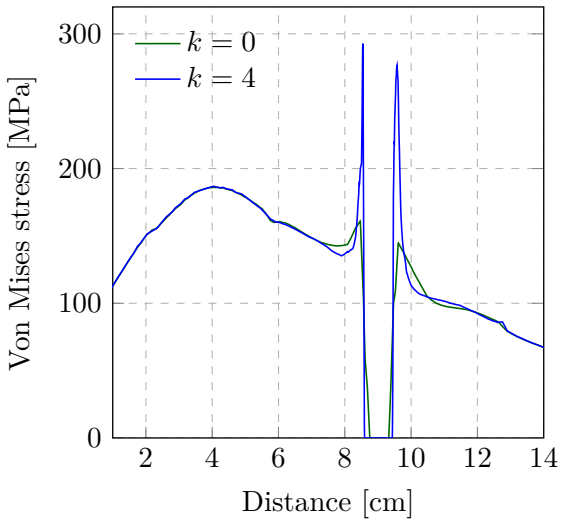
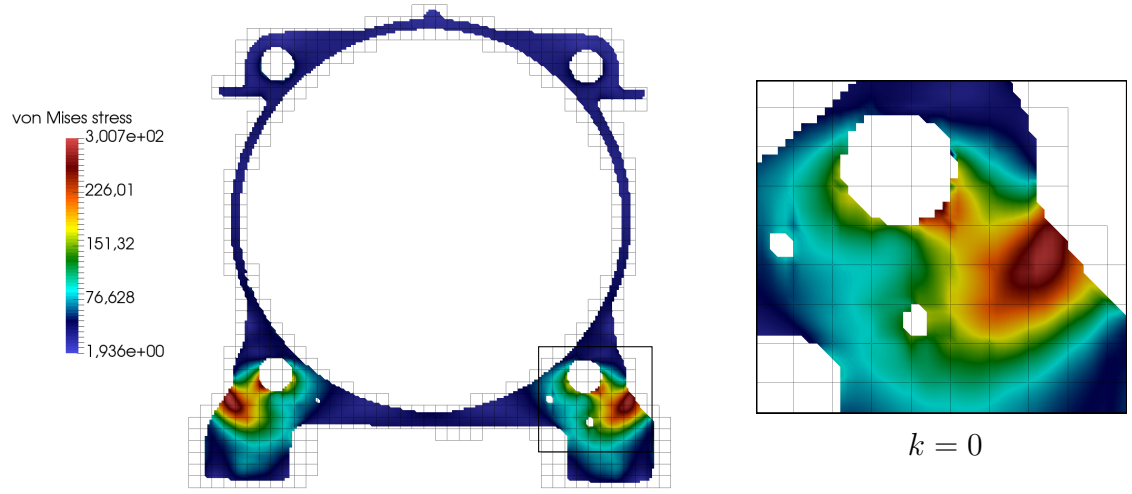
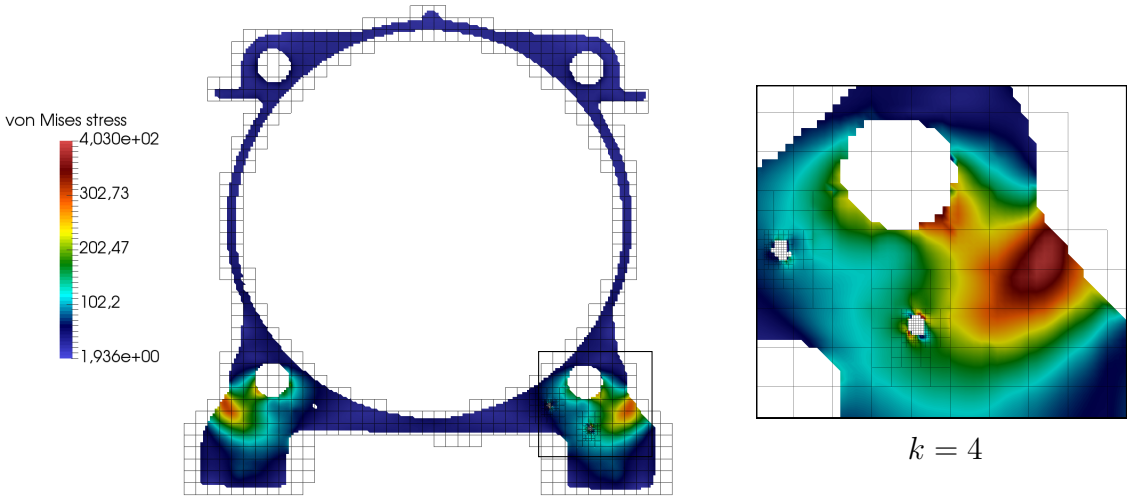


Figure 5.28: Comparison of the von Mises stress in the vicinity of the pores along the cut-line A-B shown in Figure 5.24b .



(a) Distribution of the von Mises stress in cut-plane C for $k = 0$.



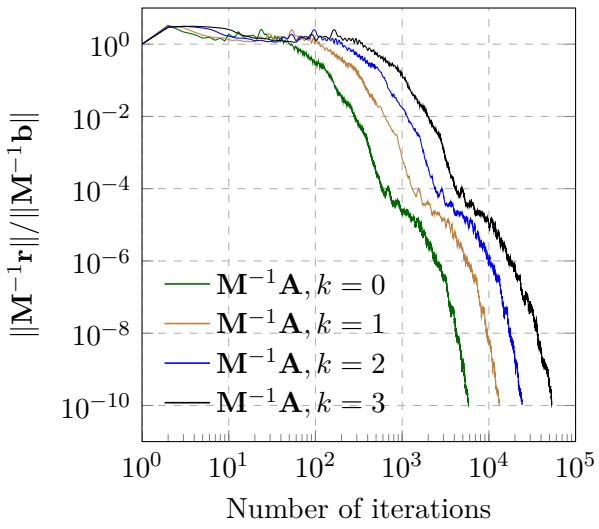
(b) Distribution of the von Mises stress in cut-plane C for $k = 4$.

Figure 5.29: Influence of the refinement depth on the stress distribution around pores in cut-plane C.

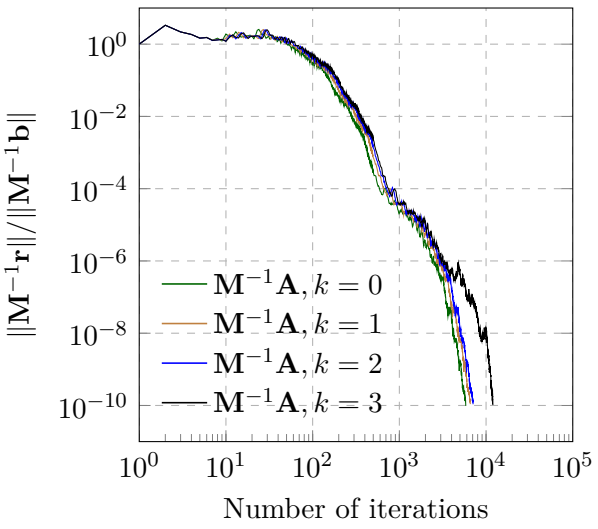
Convergence behavior

An analogous study to that in Section 5.2.5.1 is carried out to analyze the influence of multi-level *hp*-refinement on the convergence of the PCG solver. We refine the computational mesh towards the internal pores with a refinement depth $k \in \{0, 1, 2, 3\}$. The development of the relative preconditioned residual and energy error is monitored in computations involving the full and truncated preconditioners based on additive Schwarz groups formed from the leaf elements of the multi-level *hp*-grid. Figure 5.30 portrays the convergence behavior of the full and truncated preconditioner for different levels of multi-level *hp*-refinement of the gearbox housing mesh. The results show similar convergence behavior as in the numerical example considered in Section 5.2.5.1. When the full preconditioner is used, the number of iterations drastically increases with the refinement depth. The truncated preconditioner performs better

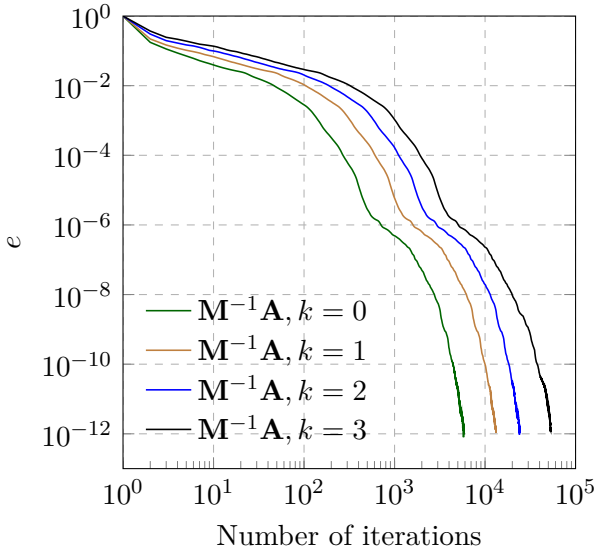
than the full preconditioner since it only takes into account basis functions that can potentially become almost linearly dependent. Table 5.3 shows the maximum number of blocks a basis function belongs to, denoted as $n_{\max}$, for the preconditioner with full and truncated blocks.



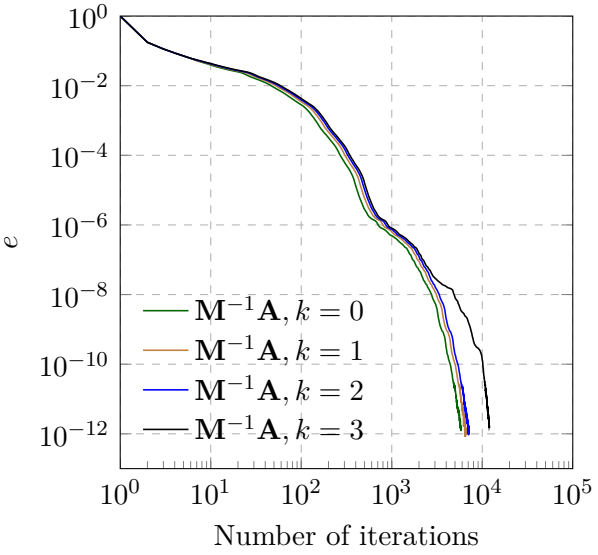
(a) Full preconditioner: residual convergence.



(b) Truncated preconditioner: residual convergence.



(c) Full preconditioner: energy convergence.



(d) Truncated preconditioner: energy convergence.

Figure 5.30: Comparison of the convergence behavior of the full and truncated preconditioners for multi-level *hp*-refinement.

k	0	1	2	3
Full blocks	8	64	274	810
Truncated blocks	8	8	8	8

Table 5.3: Maximum overlap $n_{\max}$ of the full and truncated additive Schwarz blocks for different refinement depths k in the gearbox housing example. $n_{\max}$ is bounded and equals $2^d = 8$ for the truncated preconditioner but unbounded for the full preconditioner.

Analysis of the parallel scalability

Section 5.2.4.3 describes the use of the preconditioner in a hybrid (distributed and shared memory) parallel setting. The results shown are computed using the parallelization scheme based on replicated mesh data structures described in Section 4.3. The study at hand aims to show the performance of a parallel PCG solver in large-scale finite cell computations. We use the parallel PCG solvers available in `Trilinos` [Heroux et al., 2005] in conjunction with the presented preconditioner. Two different discretizations of the gearbox housing are considered. The first discretization is the mesh used in the gearbox convergence study consisting of 34 230 elements and approximately 1.7 million degrees of freedom. The second mesh is a finer discretization of the gearbox that consists of 83 757 elements formed by grouping 7^3 voxels to form a single finite cell. A polynomial order of $p = 4$ is chosen here as well, resulting in approximately 3.98 million DOFs. Boundary conditions are applied for both discretizations as illustrated in Figure 5.25b with a penalty value $\beta = 10^6$. The numerical computations are performed on the CoolMAC cluster at the Technical University of Munich equipped with four AMD Bulldozer Opteron 6274 CPUs and 256 GB memory per node. These hybrid computations are carried out by varying the number of nodes from 1 to 16 for the coarser discretization and from 2 to 16 for the fine discretization. Each node is made up of 64 processing units (cores), i.e. one MPI-process $\times$ 64 OpenMP threads.

Figure 5.31 shows the execution time of the PCG solver plotted against the number of processing units used in the hybrid computations for both discretizations of the gearbox housing. The preconditioned solver shows excellent parallel scalability for both discretizations up to 256 cores, where superlinear speedup is achieved due to cache effects when moving to more cores. This trend can be maintained as long as the computational work in every PCG iteration — the matrix-vector multiplications — is significantly larger than the communication overhead. This is shown by the loss of efficiency in the computations involving 1.7 million degrees of freedom and more than 256 cores. The fine discretization provides adequate computational work and maintains a parallel efficiency of 115% up to 1024 processing units.

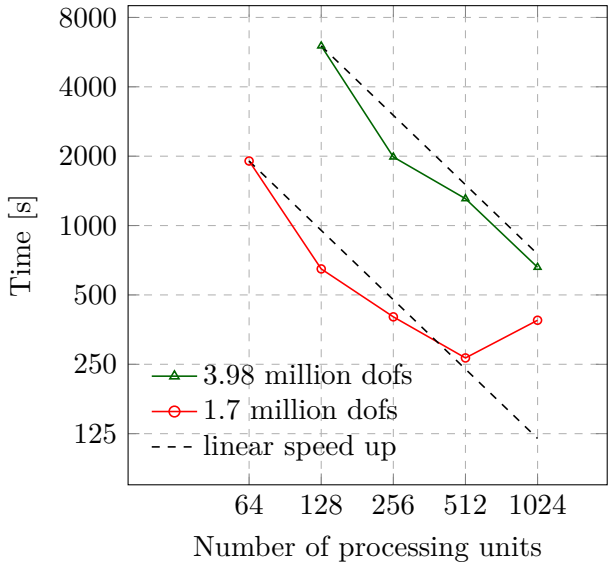


Figure 5.31: Parallel scalability study of the PCG solver for two different problem sizes.

5.3 Multigrid solution and preconditioning techniques for multi-level *hp*-FCM

Additive Schwarz preconditioners for finite cell problems involving uniform high-order meshes and multi-level *hp*-refinement were presented in the previous section. This preconditioning approach was shown to effectively treat the conditioning problems associated with badly cut cells and significantly improve the convergence of the Conjugate Gradient solver utilized. Combining the presented strategies and distributed parallelism enables the computation of large FCM systems on distributed memory machines that were previously unconceivable when a direct solver was applied.

The numerical studies in Section 5.2.3 demonstrated that finer grid sizes, higher polynomial orders and multiple levels of *hp*-refinement yield systems with larger condition numbers that generally require more CG iterations when solved. These findings are also reflected in the examples in Section 5.2.5. The bulk of the computational effort in the preconditioned Conjugate Gradient method is performing two matrix-vector products within each iteration. Each multiplication requires $\mathcal{O}(m)$ operations, with m denoting the number of nonzero entries in the sparse matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $m \in \mathcal{O}(n)$ [Shewchuk, 1994]. Following [Saad, 2003] and [de Prenter et al., 2019a] one can determine the total computational cost of a CG solver by considering the fact that the number of CG iterations required to solve a system is generally proportional to the mesh size i.e. $\mathcal{O}(h^{-1}) = \mathcal{O}(n^{\frac{1}{d}})$. The total cost can, therefore, be determined as $\mathcal{O}(n^{\frac{1}{d}}) \cdot \mathcal{O}(n) = \mathcal{O}(n^{1+\frac{1}{d}})$, where d is the dimensionality of the problem. This complexity estimate is applicable when global h-refinement is performed on meshes with a fixed polynomial order. Its derivation is based on the fact that the condition number of the stiffness matrix scales in proportion to h^{-2} in h-FEM. Deriving such an expression in the case of p-refinement is more involved. First, the growth of the condition number, a good indicator of the computational cost of solving a system, depends on the type of high-order method used

such as p-FEM with integrated Legendre polynomials, spectral elements or isogeometric analysis, see e.g. [Eisenträger et al., 2020; Maitre and Pourquier, 1996]. For the p-FEM applied in this thesis, it can be shown that the condition number scales in proportion to $p^{4(d-1)}$, where d is the dimensionality of the problem [Hu et al., 1998; Maitre and Pourquier, 1996]. Secondly, the condition number is only an indicator of the computational cost of a system and the actual iteration count is significantly influenced by the initial guess and the structure of the right-hand side. In fact, it is not uncommon to encounter scenarios in which the convergence of a system is faster for odd values of p than even values, see e.g. [Babuška et al., 1989] or *test case A* in Section 5.3.4.1.

Reducing the computational cost of solving a linear system can lead to significant improvements in terms of computational time. This can be especially beneficial when solving systems with multi-million DOFs on large distributed computers. Multigrid solution techniques provide a means of achieving a computational complexity in the order of $\mathcal{O}(n)$, i.e. the number operations needed to solve a system scales linearly with the number of unknowns. This is achieved by the use of a hierarchy of coarse discretizations that help accelerate the convergence of the finest discretization. Most multigrid-based methods are therefore characterized by convergence rates that are independent of the mesh size (h-invariance). In some methods, a convergence rate that is independent of the element polynomial order, p-invariance, is also observed.

5.3.1 Multigrid methods

A vast amount of literature exists on multigrid methods. Classical works with a more theoretical perspective include [Hackbusch, 2013; Hackbusch and Trottenberg, 1982], while [Briggs et al., 2000; Trottenberg et al., 2001; Wesseling, 2004] provide a more accessible introduction into the subject matter. The central idea behind multigrid techniques is to accelerate the convergence of a fine problem by the incorporation of correction terms generated from a hierarchy of coarse problems. These terms approximate the smooth components of the error, hereby leading to a fast convergence of smooth modes that generally converge slowly in single-grid methods. Multigrid techniques generally fall into one of two categories, *geometric* or *algebraic* methods. Geometric multigrid methods generate the coarse problems using geometrical information. In finite element analysis, this translates to the generation of coarse problems based on a sequence of discretizations with varying element sizes (h-multigrid), polynomial orders (p-multigrid) or both (hp-multigrid). Algebraic multigrid methods generate their coarse problems solely from the fine system and are usually applied in scenarios, where geometrical information is not available or cannot be easily obtained [Shapira, 2003]. In this work and in the forthcoming sections, only geometric multigrid schemes will be considered.

Multigrid methods can be applied as stand-alone solvers or as preconditioners within a Krylov method, e.g. the Conjugate Gradient method. In the latter case, the iterative method preconditioned with a multigrid technique exhibits a computational costs of $\mathcal{O}(n)$ and in many applications even outperforms multigrid as a stand-alone solver. The following section briefly explains the essential steps in a multigrid algorithm as well as the use of multigrid as a solver and preconditioner.

Ingredients of the multigrid algorithm

Multigrid methods aim at improving the convergence of a linear system by incorporating information obtained from a hierarchy of coarse discretizations. Consider a sequence of function spaces $\mathcal{V}_0, \mathcal{V}_1 \dots \mathcal{V}_{\ell-1}, \mathcal{V}_\ell$, where the index ℓ denotes the level-number and $\ell \in [0, \ell_{max}]$. $\mathcal{V}_{\ell_{max}}$ denotes the finest space which contains the solution of the original problem, whereas $\mathcal{V}_0$ denotes the coarsest space. Each multigrid level is associated with a system of equations

$$\mathbf{A}_\ell \mathbf{x}_\ell = \mathbf{b}_\ell. \tag{5.10}$$

In this work, only sequences of nested spaces are considered with the consequence that basis functions in a coarse space can be expressed as a linear combination of basis functions from a finer space. This makes it possible to define restriction operators $\mathbf{R}_\ell$ that map basis functions from the fine space of level ℓ into the next coarse space of level $\ell - 1$. Conversely, the prolongation operators $\mathbf{R}_\ell^T$ can be introduced that map a vector in the coarse space $\ell - 1$ to a vector in the fine space ℓ which corresponds to the same function. These operators allow quantities such as vectors and matrices to be transferred from one level to another e.g. $\mathbf{A}_{\ell-1} = \mathbf{R}_\ell \mathbf{A}_\ell \mathbf{R}_\ell^T$ and $\mathbf{r}_{\ell-1} = \mathbf{R}_\ell \mathbf{r}_\ell$, where r_ℓ and $r_{\ell-1}$ denote the residuals on level ℓ and $\ell - 1$, respectively.

The two main steps in a multigrid iteration are a smoothing process and the coarse-grid correction. These steps are considered complementary since they act on different components of the error. The error in each multigrid level can be expressed as the difference between the exact solution $\mathbf{x}_\ell$ and the current approximation $\tilde{\mathbf{x}}_\ell$, i.e.

$$\mathbf{e}_\ell = \mathbf{x}_\ell - \tilde{\mathbf{x}}_\ell \tag{5.11}$$

The smoothing process, or smoothing in short, efficiently reduces the oscillatory components of the error that are associated with large eigenvalues in level ℓ. Smoothing is performed by the repeated application of linear iterative methods such as fixed-point iterations. Standard smoothers include the weighted Jacobi method and the Gauss-Seidel method, but it is also possible to use fixed-point smoothing based on additive Schwarz techniques, multiplicative Schwarz techniques and incomplete LU factorizations. In this work, the smoothing process is based on additive Schwarz techniques since they can be readily applied in massively parallel settings as their construction can be done in a fully parallel manner without the need for synchronization. This is in contrast to techniques such as Gauss-Seidel and multiplicative Schwarz, which often require coordinated application in parallel settings such as multicoloring algorithms, see e.g. [Saad, 2003]. The application of the additive Schwarz smoother is done per the formula

$$\tilde{\mathbf{x}}_\ell \leftarrow \tilde{\mathbf{x}}_\ell + \omega \mathbf{M}_\ell^{-1} \mathbf{r}_\ell, \tag{5.12}$$

where ω denotes the relaxation parameter and $\mathbf{M}^{-1}$ the additive Schwarz smoother. Note that this operation can be performed by explicitly constructing the matrix $\mathbf{M}^{-1}$ as per Algorithm 5.1, or in a matrix-free manner following (5.5). The choice of the relaxation parameter ω for the various FCM meshes applied in this manuscript is elaborated in Section 5.3.3.

The coarse-grid correction accelerates the convergence of the smooth components of the error that are associated with small eigenvalues in level ℓ. By taking advantage of the relation between the residual $\mathbf{r}_\ell$ and the error $\mathbf{e}_\ell$

$$\mathbf{A}_\ell \mathbf{e}_\ell = \mathbf{r}_\ell \tag{5.13}$$

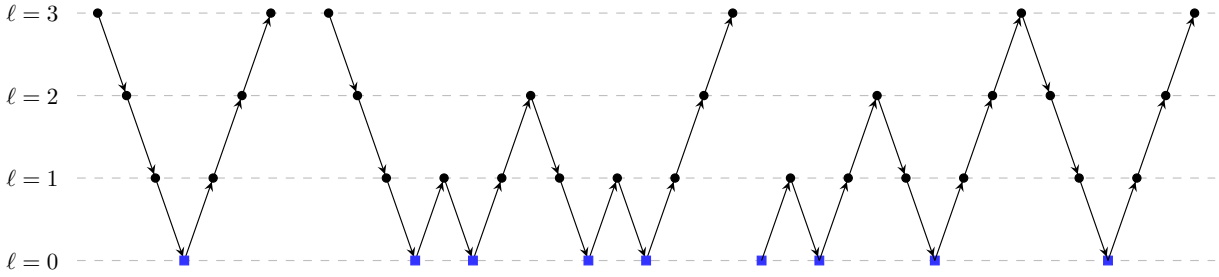
it is possible to approximate the smooth components of the error $\mathbf{e}_\ell$ using a coarser discretization (grid), where $\mathbf{e}_\ell \approx \mathbf{R}_\ell^T \mathbf{e}_{\ell-1}$, by solving the coarse-grid system

$$\mathbf{A}_{\ell-1}\mathbf{e}_{\ell-1} = \mathbf{r}_{\ell-1}. \tag{5.14}$$

The right-hand side vector of this coarse system is formed by restricting the residual at level ℓ to level $\ell - 1$, i.e. $\mathbf{r}_{\ell-1} = \mathbf{R}_\ell \mathbf{r}_\ell$. Once this system has been solved, its solution termed the coarse-grid correction can be transferred to level ℓ through the expression

$$\tilde{\mathbf{x}}_\ell = \tilde{\mathbf{x}}_\ell + \mathbf{R}_\ell^T \mathbf{e}_{\ell-1}. \tag{5.15}$$

A single multigrid iteration or cycle consists of the repeated application of the smoothing and coarse-grid correction steps on the different multigrid levels. An exception is, however, made on the coarsest level $\ell = 0$. Here, the coarse system is solved with either a direct solver, if the system is small in size, or with an iterative solver before prolongating its solution to level $\ell = 1$. There are different ways of traversing the levels within a multigrid cycle, the most prevalent of which are the V-cycle, W-cycle and the full multigrid cycle (FMG). A schematic representation of each of these cycles is given in Figure 5.32. V-cycles are applied in this work and a summary of the multigrid algorithm for this iteration type is given in Algorithm 5.2.



(a) V-cycle. (b) W-cycle. (c) FMG-cycle.

Figure 5.32: Illustration of three different types of multigrid iterations. The circular dots represent n_s smoothing steps performed on every level with $\ell \neq 0$, while the square dots represent an "exact" solve performed on $\ell = 0$. Downwards pointing arrows represent restriction operations, while upwards pointing arrows represent prolongations.

Multigrid as a solver

Applying the multigrid algorithm as a solver entails the repetition of multigrid cycles until the desired error tolerance of the finest problem is reached. This process is summarized for the V-cycle in Algorithm 5.3. One quantity that is often used to judge the effectiveness of a multigrid solver is the contraction number ρ_i. It is defined as the quotient of two consecutive residual norms and is a measure of the error reduction in the iteration process. ρ_i is computed as

$$\rho_i = \frac{\|\mathbf{r}_i\|_2}{\|\mathbf{r}_{i-1}\|_2}. \tag{5.16}$$

Note that the subscripts in the above equation refer to multigrid iterations as shown in Algorithm 5.3. ρ_{max} is a measure that denotes the maximum contraction number in a series of iterations.

Multigrid as a preconditioner

Multigrid methods can be used to accelerate the convergence of Krylov space solvers. Early research in this field can be found in [Braess, 1986; Braess and Peisker, 1986; Kettler, 1982]. Braess [1986] reports that multigrid as a preconditioner shows improved convergence in scenarios where multigrid solver's contraction number $\rho_{\max}$ is larger than 1/3. Algorithm 5.4 shows how the multigrid V-cycle can be used within a CG solver. In this thesis, a single multigrid V-cycle is applied as a preconditioner in each iteration. When applying a multigrid method in a CG solver, it is important that the multigrid cycle corresponds to the application of a symmetric preconditioner. For this reason, nonsymmetric smoothers such as Gauss-Seidel, SOR and multiplicative Schwarz have to be replaced by their symmetric pre- and post-smoothing versions.

Algorithm 5.2: $\tilde{\mathbf{x}}_\ell = \text{performVcycle}(\tilde{\mathbf{x}}_\ell, \mathbf{r}_\ell, \ell)$

if $l \neq 0$ **then**
  # perform n_s pre-smoothing steps
  for $i \in n_s$ **do**
    $\tilde{\mathbf{x}}_\ell \leftarrow \tilde{\mathbf{x}}_\ell + \omega \mathbf{M}_\ell^{-1} \mathbf{r}_\ell$
  end
  # update residual
  $\mathbf{r}_\ell = \mathbf{b}_\ell - \mathbf{A}_\ell \tilde{\mathbf{x}}_\ell$
  # coarse grid correction
  $\mathbf{r}_{\ell-1} = \mathbf{R}_\ell \mathbf{r}_\ell$
  $\tilde{\mathbf{x}}_{\ell-1} = \text{performVcycle}(\mathbf{0}, \mathbf{r}_{\ell-1}, \ell - 1)$
  $\tilde{\mathbf{x}}_\ell = \tilde{\mathbf{x}}_\ell + \mathbf{R}_\ell^T \tilde{\mathbf{x}}_{\ell-1}$
  $\mathbf{r}_\ell = \mathbf{b}_\ell - \mathbf{A}_\ell \tilde{\mathbf{x}}_\ell$
  # perform n_s post-smoothing steps
  for $i \in n_s$ **do**
    $\tilde{\mathbf{x}}_\ell \leftarrow \tilde{\mathbf{x}}_\ell + \omega \mathbf{M}_\ell^{-1} \mathbf{r}_\ell$
  end
else
  # solve the coarse system
  $\tilde{\mathbf{x}}_\ell = \text{solve}(\mathbf{A}_\ell, \mathbf{r}_\ell)$
end

Algorithm 5.3: $\tilde{\mathbf{x}} = \text{multigridSolver}(\mathbf{A}, \mathbf{b})$

$\mathbf{x} = \mathbf{0}$ # initial guess
$\mathbf{r} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}$
# perform n_{cycles} consecutive V-cycles
for $i \in n_{cycles}$ **do**
  $\mathbf{r} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}$
  $\mathbf{x} = \text{performVcycle}(\tilde{\mathbf{x}}, \mathbf{r}, \ell_{max})$
  if $\|r\|/\|r_0\| < tol$ **then**
    break;
  end
end

Algorithm 5.4: $\tilde{\mathbf{x}} = \text{PCGMGSolver}(\mathbf{A}, \mathbf{b})$

$\mathbf{x} = \mathbf{0}$ # initial guess usually $\mathbf{0}$
$\mathbf{p}_0 = \mathbf{0},\ \mathbf{z}_0 = \mathbf{0}$
$\mathbf{r_0} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}$
$\mathbf{z_0} = \text{performVcycle}(\mathbf{r}_0, \mathbf{z}_0, \ell_{max})$
# perform n_{iters} iterations
for $k \in n_{iters}$ **do**
  $\alpha_k = \dfrac{\mathbf{r}_k^T \mathbf{z}_k^T}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k}$
  $\tilde{\mathbf{x}}_{k+1} = \tilde{\mathbf{x}}_k + \alpha_k \mathbf{p}_k$
  $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$
  if $\|r\|/\|r_0\| < tol$ **then**
    break;
  end
  $\mathbf{z}_{k+1} = \text{performVcycle}(\mathbf{0}, \mathbf{r}_{k+1}, \ell_{max})$
  $\beta_k = \dfrac{\mathbf{z}_{k+1}^T \mathbf{r}_{k+1}^T}{\mathbf{z}_k^T \mathbf{r}_k}$
  $\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k$
end

5.3.2 A *hp*-multigrid approach for the multi-level *hp*-method

Multigrid methods have been successfully applied to boundary conforming FE-methods based on the *p*-version of the finite element method e.g. [Babuška et al., 1989; Craig and Zienkiewicz, 1985] and isogeometric analysis e.g. [Gahalaut et al., 2013; Hofreither et al., 2016; Tielen et al., 2020], yielding convergence rates that are independent of the grid-size h. *p*-invariance of the iteration count is, however, harder to achieve and is dependent on the type of smoother employed and the problem type. In most cases, standard smoothers such as Jacobi or Gauss-

Seidel result in convergence rates that deteriorate with increasing values of p, as shown in [Gahalaut et al., 2013]. In [Tielen et al., 2020], it is shown that using an ILUT smoother in an isogeometric p-multigrid framework helps achieve p-independence for the problem classes considered therein. Likewise, smoothers based on the multiplicative Schwarz algorithm have been shown to yield convergence rates independent of p for two-dimensional problems in isogeometric analysis [de la Riva et al., 2019].

Approaches based on multigrid cycles have also been employed in immersed methods. An algebraic multigrid solver is utilized in AgFEM to solve large scale problems in the range of hundreds of millions of unknowns [Verdugo et al., 2019]. In [de Prenter et al., 2019b] a h-multigrid approach for truncated hierarchical B-splines is presented and convergence rates that are independent of the cut configuration and the element size are reported. An analysis of the spectra arising from the application of different smoothers is given by de Prenter et al. and results based on multiplicative Schwarz smoothing of immersed systems with up to 10 million unknowns are shown.

One possibility of constructing the nested multigrid spaces in high-order and adaptive finite element methods is by defining them based on the values of p, the different levels of refinements k, or a combination of p and k. Selecting the spaces in this way gives rise to so-called p-multigrid and hp-multigrid techniques. Several works exist on the analysis and use of these methods such as [Babuška et al., 1989; Craig and Zienkiewicz, 1985; Foresti et al., 1989; Yserentant, 1985, 1986]. A comprehensive review of the origin and development of these methods can be found in [Mitchell, 2010].

In this work, a multilevel multigrid method is proposed that takes advantage of the hierarchical nature of the finite element spaces arising in FCM and the multi-level hp-refinement scheme. As previously mentioned, FCM utilizes high-order integrated Legendre shape functions that are hierarchical in nature, i.e. the set of basis functions of order p contains all basis functions from 1 up to $p-1$. Likewise, the superposition principle used to perform h-refinements in the multi-level hp-method results in a hierarchical basis. Let $\mathbf{N}_\ell$ denote the set of basis functions that belong to the multigrid level ℓ, then for each level in a p-multigrid (for uniform meshes) or hp-multigrid (for multi-level hp-meshes) the following relation holds

$$\mathbf{N}_0 \subset \mathbf{N}_1 \ldots \mathbf{N}_{\ell-1} \subset \mathbf{N}_\ell. \tag{5.17}$$

The hierarchical structure in (5.17) is also reflected in the degree of freedom vector $\mathbf{u}$. The vector of DOFs on level ℓ, denoted by $\mathbf{u}_\ell$, is made up of coefficients from level $\ell-1$, represented by $\mathbf{u}_{\ell-1}$ and entries solely on level ℓ, denoted by w_ℓ, i.e.

$$\mathbf{u}_\ell = \begin{bmatrix} \mathbf{u}_{\ell-1} \\ \mathbf{w}_\ell \end{bmatrix}. \tag{5.18}$$

From (5.17) it follows that the basis functions spanning the (coarse) nested subspace can not only be constructed by a linear combination of basis functions in the fine space, which is done in restriction and prolongation, but are explicitly contained in the basis functions of the fine space. This leads to an elegant and efficient multigrid framework since all restriction and prolongation operators reduce to binary matrices which do not need to be explicitly applied. Transitioning from one multigrid level to another is done easily by either leaving out specific basis functions or re-introducing them back into the system. Equation (5.19) illustrates how

the DOFs on level ℓ can be "trimmed" to obtain the DOFs on level $\ell - 1$. $\mathbf{I}$ represents a identity matrix while the term $\mathbf{0}$ denotes a matrix in which all entries are zero.

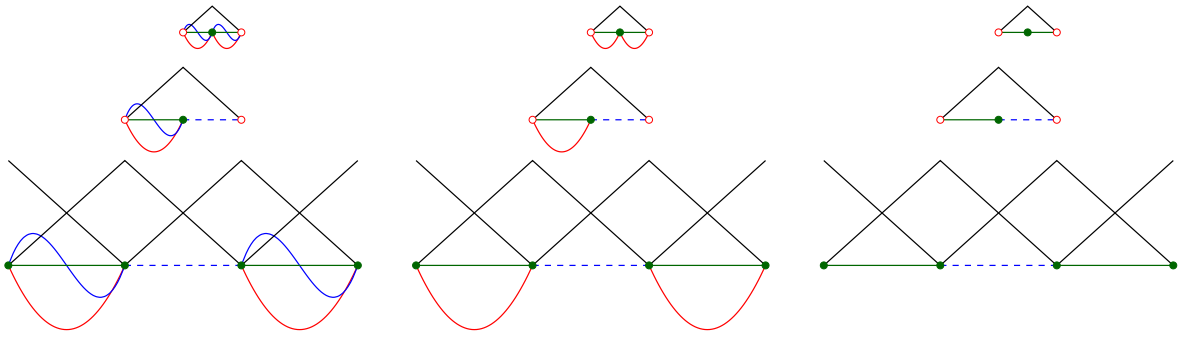
$$\mathbf{u}_{\ell-1} = [\mathbf{I},\ \mathbf{0}] \begin{bmatrix} \mathbf{u}_{\ell-1} \\ \mathbf{w}_{\ell} \end{bmatrix}. \tag{5.19}$$

For uniform grids with high-order elements, an *arithmetic p-sequence* is used to generate the coarse subspaces, i.e. the value of p is progressively reduced until $p = 1$, see Mitchell [2010]. In the case of multi-level *hp*-grids, it is required to first reduce the polynomial order p in the overlay meshes, before reducing the levels of refinement. This procedure ensures that the resulting coarse spaces are subspaces.

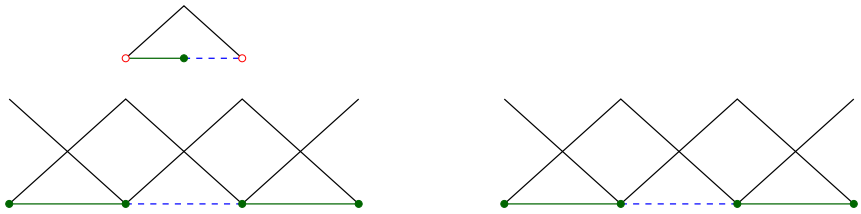
The hierarchical nature of the FE-basis in FCM and the multi-level *hp*-scheme is also reflected in the system matrices. Consequently, computation of lower-level matrices is not needed as this information is readily available. Equation (5.20) shows the structure of a hierarchical matrix of level ℓ, which consists of entries $\tilde{\mathbf{A}}_{\ell}$ belonging to basis functions contained solely in the highest level, a term $\tilde{\mathbf{A}}_{\ell-1}$ that contains entries of all lower levels up to $\ell - 1$ and a term $\tilde{\mathbf{A}}_{\ell,\ell-1}$ that couples DOFs on level ℓ with all other DOFs.

$$\mathbf{A}_{\ell} = \begin{bmatrix} \tilde{\mathbf{A}}_{\ell} & \tilde{\mathbf{A}}_{\ell,\ell-1} \\ \tilde{\mathbf{A}}^{T}_{\ell,\ell-1} & \mathbf{A}_{\ell-1} \end{bmatrix} \tag{5.20}$$

No distinction will be made from this point on in the manuscript between the *p*-multigrid scheme for uniform meshes and the *hp*-multigrid approach of the multi-level *hp*-discretizations, since the *p*-multigrid can be regarded as a special case of the *hp*-multigrid in which $k = 0$. Figure 5.33 shows the *hp*-multigrid levels used for a one-dimensional multi-level *hp*-mesh. A reduction in the *p*-level is performed first, followed by a reduction in the levels of refinement until the lowest multigrid level with $p = 1$ and $k = 0$ is reached.



(a) $\ell = 4$ with $p = 3$ and $k = 2$ (b) $\ell = 3$ with $p = 2$ and $k = 2$ (c) $\ell = 2$ with $p = 1$ and $k = 2$



(d) $\ell = 1$ with $p = 1$ and $k = 1$ (e) $\ell = 0$ with $p = 1$ and $k = 0$

Figure 5.33: Multigrid levels in a one-dimensional mesh with two levels of multi-level *hp*-refinement ($k = 2$) and a polynomial order of $p = 3$.

Remark 5.3.1. *The $p = 1$ and $k = 0$ level is chosen as the coarsest level in the proposed hp-multigrid approach. This allows a simple yet elegant parallel implementation, since only the DOFs on the finest problem need to be distributed over the MPI tasks. All coarse sub-problems in the multigrid hierarchy are able to reuse the parallel data structures that have already been set up for the fine problem, and do not require any additional steps such as redistribution or partitioning of the DOFs. It should be noted, however, that the $p = 1, k = 0$ problem can be coarsened further using a standard geometric multigrid algorithm. This procedure can improve the convergence of the coarse problem and is beneficial in applications where the solution of the coarse problem is slow.*

5.3.3 Selection of suitable smoothing strategies

The choice of an appropriate smoother is integral in achieving convergence rates that are independent of the mesh parameter h. In specific problems, it is even possible to achieve convergence rates that are independent of the element polynomial order, such as those reported in [Tielen et al., 2020] or handled in Section 5.3.4.1 and 5.3.4.2.

When the multi-level *hp*-refinement is applied to boundary-conforming discretizations, standard smoothing techniques such as Jacobi smoothing or the Gauss-Seidel can be applied, as both these approaches lead to convergence rates independent of h, see Section 5.3.4.1. These techniques are, however, not suitable for FCM problems as shown in [de Prenter et al., 2019b] as they fail to resolve the conditioning problems associated with cut cells. [de Prenter et al., 2019b] shows that smoothers based on the additive and multiplicative Schwarz lemmas are better suited for multigrid methods involving cut cells and presents results of multiplicative Schwarz smoothing in different linear elastic examples.

As mentioned in the previous section, an additive Schwarz approach is applied in this work since it is less computationally expensive than the multiplicative approach, and can be easily parallelized as it does not require a coordinated application. This smoothing approach, however, requires sufficient stabilization in order to converge. It is well known that the convergence of fixed-point iterations requires that the eigenvalues of the iteration matrix are bounded, i.e.

$$\rho(\mathbf{I} - \omega \mathbf{M}^{-1}\mathbf{A}) < 1. \tag{5.21}$$

In [de Prenter et al., 2019b], it is shown that the largest eigenvalue of the matrix $\mathbf{M}^{-1}\mathbf{A}$ is bounded from above by the maximum overlap $n_{\max}$ of the additive Schwarz blocks i.e. $\lambda_{\max}(\mathbf{M}^{-1}\mathbf{A}) \leq n_{\max}$. Since the value of n_{max} is 2^d for Cartesian grids, the relaxation parameter ω can be chosen as $\omega = 2/n_{\max}$, thus guaranteeing stability of the fixed-point iteration. Note, however, that the relation between the largest eigenvalue and $n_{\max}$ is an inequality and that it is possible to choose values for ω that are higher than $2/n_{\max}$. For the integrated Legendre basis functions considered in this work, the best performance was achieved by choosing $\omega = 1/3$ for two-dimensional problems and $\omega = 2/15$ for three-dimensional problems, when using an additive Schwarz smoother based on elementwise blocks. When the additive Schwarz blocks are chosen in a patchwise manner $n_{\max}$ is larger and the best convergence rates are achieved using $\omega = 1/6$ and $\omega = 1/18$ in two and three dimensions respectively. The values of ω suggested in this work were determined in a heuristic approach and achieve the mesh independent convergence rates for different geometries.

It should be noted that different smoothing strategies can be applied on the different multigrid levels, e.g. additive Schwarz on the finest level and Gauss-Seidel or Jacobi smoothing on lower levels. This is, however, not investigated in this thesis and additive Schwarz smoothing is applied on all levels with $\ell \neq 0$.

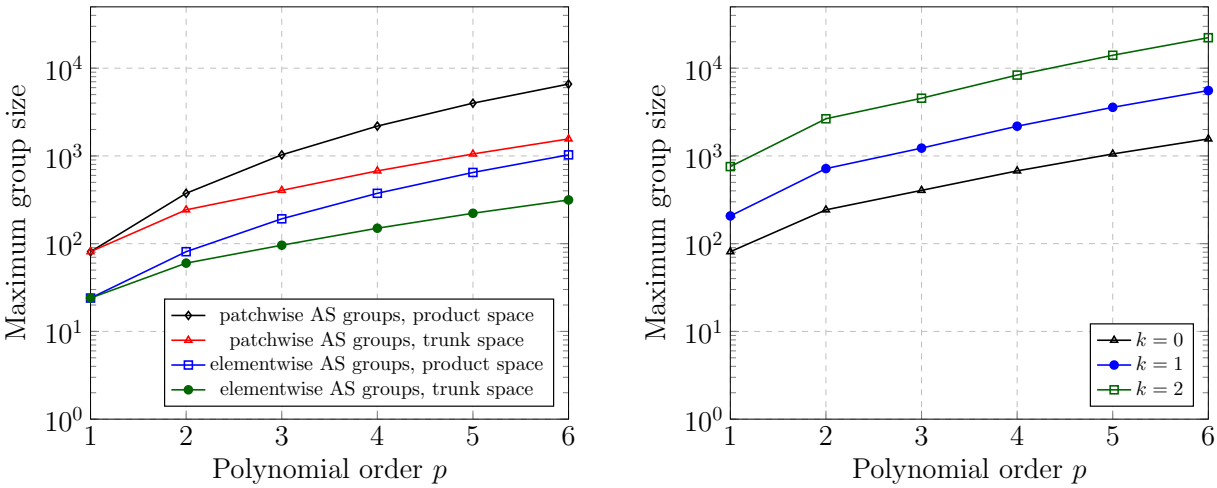
Computational costs

The two main procedures that influence the overall computational cost of the proposed multigrid solution scheme are: *i)* constructing and applying the additive Schwarz smoothers and *ii)* solving the coarse problem.
Construction of the AS smoothers following (5.1) entails the inversion of sub-matrices derived from basis function groups. The cost of this operation is dictated by the number and size of these sub-matrices. The number of sub-matrices increases linearly with the number of elements. The maximum size of a sub-matrix formed from the additive Schwarz groups, denoted by $m_{\max}$, is proportional to the polynomial order p, the spatial dimension d, the manner in which the AS groups are selected (elementwise or patchwise selection) and the number of unknown field variables denoted by n_f. For a Poisson problem $n_f = 1$ while $n_f = 3$ for a three-dimensional linear elastic problem. Since the number of DOFs associated with the topological components for the tensor product and trunk space elements is known, it is possible to compute an upper bound for $m_{\max}$ for uniform grids, see Table 5.4 and Figure 5.34a.

tensor product space	**trunk space**
$n_f(np+1)^3$	$n_f(n+1)^2(3np-2n+1)$ for $p<4$
	$0.5n_f(n+1)(3n^2p^2-9n^2p+6np+14n^2-2n+2)$ for $4\leq p\leq 5$
	$n_f(n^3p^3-3n^3p^2+9n^2p^2+20n^3p-9n^2p+18(np-n^3+2n^2)+6)$ for $p\geq 6$

Table 5.4: Maximum size of the additive Schwarz groups, $m_{\max}$, in a uniform three-dimensional grid. p represents the element polynomial order, n_f the number of field variables in the problem and n is a factor that is equal to one for elementwise blocks and equal to two when patchwise block selection is applied.

In the case of multi-level *hp*-grids, $m_{\max}$ is not bounded and its size depends on the refinement level k and the refinement pattern applied to the mesh elements. Figure 5.34b shows the value of $m_{\max}$ for the benchmark problem considered in Section 5.3.4.3. From Table 5.4 and Figure 5.34 it is clear that the inversion of the patchwise additive Schwarz sub-matrices can become increasingly expensive for high polynomial orders and refinement levels. It is therefore important that optimized algorithms are used to perform these inversions. Our code framework makes use of distributed and shared memory parallelism to accelerate the construction of the additive Schwarz smoothers. For "small" sub-matrices, where $\mathbf{A}_i \in \mathbb{R}^{m\times m}$ and $m < 1000$, the built-in invert function of the `BOOST` library is used for the inversion. When m exceeds 1000, the direct solver `Pardiso` [Schenk and Gärtner, 2011] is used to invert $\mathbf{A}_i$ by solving an equation system with m right-hand side vectors. This operation can be written as $\mathbf{A}_i\mathbf{X} = \mathbf{B}$, where $\mathbf{B}$ is a $m \times m$ matrix whose columns are made up of the unit vectors $\mathbf{e}_j$ with $j \in [0, m]$.



(a) Maximum AS group size for uniform grids.

(b) Maximum group size for the benchmark example involving multi-level *hp*-refinement in Section 5.3.4.3.

Figure 5.34: Illustration of the maximum size of the basis function groups used to construct the additive Schwarz smoothers for three-dimensional linear elastic problems.

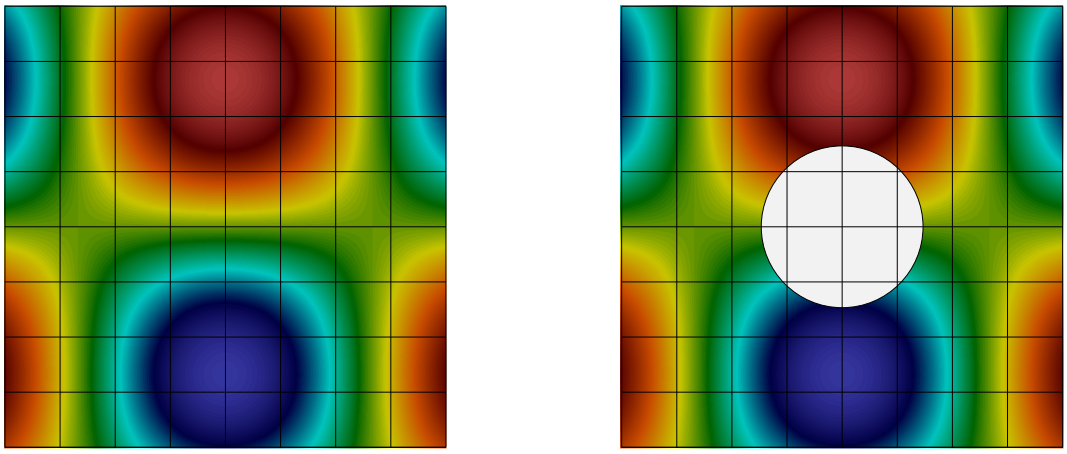
The use of the `Epetra` package for the distributed storage of the additive Schwarz smoothers $\mathbf{M}_\ell^{-1}$ allows the numerical code presented in this thesis to make use of its optimized parallel multiplication kernels. This ensures that the smoothers can be applied in an efficient and scalable manner. The `AztecOO` package in `Trilinos` is utilized for the solution of the coarse systems with $p = 1$ and $k = 0$. These systems are solved using the package's parallel CG solver and an additive Schwarz preconditioner.

5.3.4 Numerical examples

The performance of the proposed multigrid solution techniques are investigated in the following numerical examples. Only SPD systems arising from problems in linear elasticity or Poisson's equation are considered in this section. A value $n_s = 5$ is used in all examples, meaning that five pre- and post-smoothing steps are applied on every multigrid level $\ell \neq 0$ in a V-cycle.

5.3.4.1 Poisson problem with a manufactured solution

The first numerical examples investigates the performance of using the multigrid V-cycle as a stand-alone solver and as a preconditioner within a Conjugate Gradient scheme. To this end, the Poisson problem with a manufactured solution handled in Section 5.2.1 is applied on two simple two-dimensional geometries that are depicted in Figure 5.35 . The first geometry is a square-shaped domain of unit length, that shall be used to test the implementation of the multigrid solvers when applied to boundary conforming discretizations in a series of numerical experiments referred to as *test case A*. The second geometry constitutes a unit square with a circular cavity of a radius $r = \sqrt{2}/8$ at its center. This domain is used to assess the solver performance for FCM grids in studies referred to as *test case B*. A fixed penalty parameter $\beta = 10^5$ is used in all experiments and the value of α is set to 10^{-6}.



(a) Square domain — Test case A. (b) Square domain — Test case B.

Figure 5.35: Discretizations considered in test case A and B.

In both test cases A and B, the convergence behavior of multigrid as a solver and a preconditioner is analyzed for different polynomial orders with $p \in [2, \dots, 5]$ and mesh sizes $h = \{\frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}\}$. Each solver is terminated when the value of the relative residual is below 10^{-9} or when the number of iterations exceeds 500. Moreover, four different smoothers are applied: *i)* Jacobi smoothing, *ii)* Gauss-Seidel smoothing, *iii)* additive Schwarz smoothing with elementwise blocks and *iv)* additive Schwarz smoothing with patchwise blocks.

The results of test case A are summarized in Table 5.7. In this boundary-conforming example, all smoothers achieve convergence rates that are independent of the mesh size h. These results indicate the correct implementation of the multigrid scheme. These convergence rates are, however, dependent on the polynomial order for the Jacobi, Gauss-Seidel and elementwise additive Schwarz smoothers as shown in Table 5.7 and indicated by the maximum contraction of the residual for a mesh with $h = \frac{1}{32}$ recorded in Table 5.5. These smoothers appear to perform better for odd polynomial orders than for even orders. A similar odd-even pattern is reported in [Babuška et al., 1989] and may lie in the asymmetry of the manufactured solution. Convergence rates independent of the polynomial order p are obtained using the patchwise additive Schwarz smoother.

Smoother \ p	2	3	4	5
Jacobi	.478	.391	.683	.617
Gauss-Seidel	.125	.095	.362	.255
Elementwise AS	.309	.252	.430	.414
Patchwise AS	.162	.164	.140	.162

Table 5.5: Maximum contraction number $\rho_{\max}$ of multigrid as a solver for different values of p. The values are obtained in test case A for a mesh with $h = \frac{1}{32}$.

Smoother \ p	2	3	4	5
Jacobi	.996	.997	.999	.998
Gauss-Seidel	.998	.998	.998	*
Elementwise AS	.519	.628	.734	.891
Patchwise AS	.126	.103	.124	.119

Table 5.6: Maximum contraction number $\rho_{\max}$ of multigrid as a solver for different values of p. The values are obtained in test case B for a mesh with $h = \frac{1}{32}$.

The results of test case B are summarized in Table 5.8. The effect of the ill-conditioning due to the cut cells is clearly seen in this example as the standard smoothers, i.e. the Jacobi and Gauss-Seidel methods, fail to improve the conditioning of the linear systems and are characterized by poor convergence of the multigrid solution techniques applied in this study. An analysis of the spectra of the multigrid preconditioned system clearly shows that both Jacobi and Gauss-Seidel smoothing fail to act on the small eigenvalues associated with cut cells, see Figure 5.36. The additive Schwarz smoothers do a better job of detecting almost linear dependent functions as shown in Table 5.8. Note that the elementwise smoother may contain a few small modes that cause slow convergence when multigrid is utilized as a stand-alone solver. These modes, however, only result in a few additional CG iterations, when the multigrid algorithm is used as a preconditioner. The patchwise smoother robustly deals with all small modes due to cut cells and results in convergence rates independent of h and p.

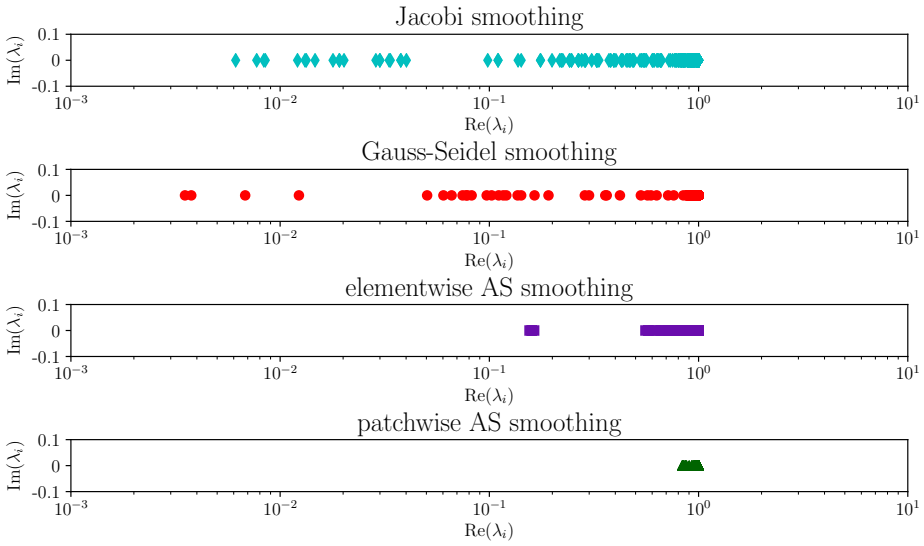


Figure 5.36: Comparison of the eigenvalues of the multigrid preconditioned systems for four different smoothers i.e. a Jacobi smoother, a Gauss-Seidel smoother, an elementwise additive Schwarz smoother and a patchwise additive Schwarz smoother. The spectra shown belong to a mesh comprising of 8×8 elements with a polynomial order $p = 2$.

Test case A: Convergence study for a square domain

	Multigrid as a solver Jacobi smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	23	25	25	24
3	17	16	15	15
4	31	33	35	37
5	24	24	26	27

	CG with multigrid preconditioner Jacobi smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	10	10	10	10
3	8	8	8	8
4	13	13	13	13
5	10	11	11	11

	Multigrid as a solver Gauss-Seidel smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	7	7	7	7
3	6	6	6	6
4	13	13	14	14
5	10	10	10	10

	CG with multigrid preconditioner symm. Gauss-Seidel smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	6	6	6	6
3	5	5	5	5
4	8	8	8	8
5	8	7	7	7

	Multigrid as a solver elementwise AS smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	12	13	13	13
3	10	10	10	9
4	17	17	17	17
5	14	12	13	13

	CG with multigrid preconditioner elementwise AS smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	7	8	8	8
3	7	7	7	7
4	9	9	9	9
5	8	8	8	8

	Multigrid as a solver patchwise AS smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	8	8	8	7
3	7	6	6	6
4	7	7	7	6
5	6	6	5	5

	CG with multigrid preconditioner patchwise AS smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	6	6	6	6
3	6	5	5	5
4	6	6	6	6
5	5	5	5	5

Table 5.7: Convergence study for $\psi = 0$ comparing the performance of different smoothers for varying element sizes and polynomial orders. The figures in the tables represent the number of iterations required to reach a tolerance in the relative residual of 10^{-9}. Five pre- and post-smoothing steps are performed per V-cycle on each multigrid level with $\ell \neq 0$.

Test Case B: Convergence study for a square domain with a circular cavity

	Multigrid as a solver Jacobi smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	*	*	*	*
3	*	*	*	*
4	*	*	*	*
5	*	*	*	*

	CG with multigrid preconditioner Jacobi smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	36	75	125	189
3	67	340	*	*
4	167	*	*	*
5	*	*	*	*

	Multigrid as a solver Gauss-Seidel smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	*	*	*	*
3	*	*	*	*
4	*	*	*	*
5	*	*	*	*

	CG with multigrid preconditioner Gauss-Seidel smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	46	95	188	224
3	106	435	*	*
4	216	*	*	*
5	401	*	*	*

	Multigrid as a solver elementwise AS smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	12	18	12	13
3	134	58	13	12
4	*	*	30	21
5	*	*	101	20

	CG with multigrid preconditioner elementwise AS smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	8	10	9	9
3	9	11	10	9
4	14	13	11	10
5	16	14	11	10

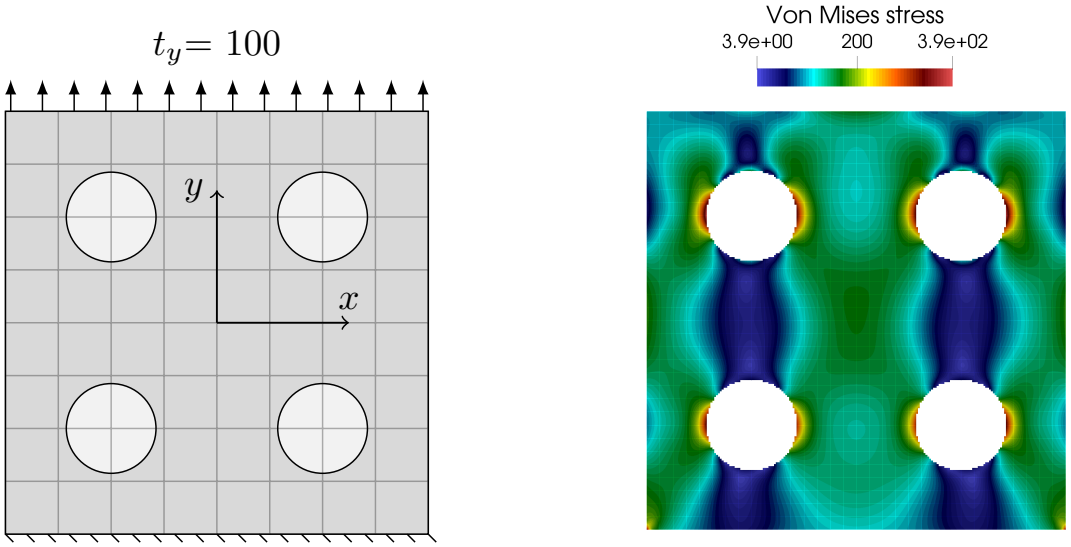
	Multigrid as a solver patchwise AS smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	8	9	7	7
3	8	7	7	6
4	7	7	7	6
5	6	6	8	9

	CG with multigrid preconditioner patchwise AS smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	5	5	5	5
3	5	5	5	4
4	5	5	5	4
5	4	5	5	4

Table 5.8: Convergence study for $\psi = 30°$ comparing the performance of different smoothers for varying element sizes and polynomial orders. The figures in the tables represent the number of iterations required by the solver while the symbol * indicates that the solver did not converge to a tolerance of 10^{-9} within 500 iterations. Five pre- and post-smoothing steps are performed per V-cycle on each multigrid level with $\ell \neq 0$

5.3.4.2 Perforated linear elastic plate

The previous numerical example established the suitability of using a multigrid preconditioner in conjunction with additive Schwarz smoothers in FCM problems arising from the Poisson equation. The performance of this preconditioner is now investigated in the context of linear elasticity. To this end, a square domain with a length of $l = 4$ is subjected to a tensional force on one end and fully clamped on the other end as depicted in Figure 5.37a. The square domain has four circular cavities with a radius of $0.3\sqrt{2}$ and is characterized by an elastic modulus $E = 2.069 \cdot 10^5$ MPa and a Poisson's ratio $\nu = 0.29$. The finite cell method is applied in this example with $\alpha = 10^{-8}$ in Ω_{fict} and a penalty parameter $\beta = 10^8$. The number of elements per direction is chosen such that $h \in \{\frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}\}$.



(a) Perforated linear elastic plate

(b) Von Mises stress.

Figure 5.37: Setup and von Mises stress of the linear elastic perforated plate example.

Using the setup shown in Figure 5.37a, numerical studies are carried out that investigate the convergence of a CG solver when the presented additive Schwarz techniques are employed as preconditioners or as smoothers within an *hp*-multigrid preconditioner. The first study is conducted on uniform grids with varying element sizes and polynomial orders, while the second study analyzes multi-level *hp*-refined grids with a fixed polynomial order and varying levels of refinement. In the latter case, elements intersected by the immersed boundary are refined recursively to a predefined depth.

The results of the numerical study on uniform grids is summarized in Table 5.9. Faster convergence is achieved for the systems that use a multigrid preconditioner compared to the systems where additive Schwarz preconditioning is employed. The two coarsest meshes with $h = \frac{1}{8}$ and $h = \frac{1}{16}$ have the peculiarity that their rather large elements lead to the occurrence of basis functions at the center of the circular cavities that connect disjoint parts of the physical domain, see Figure 5.37a. These functions are not treated by the elementwise additive Schwarz blocks and cause slow convergence of the Conjugate Gradient solvers that employ this technique as either a smoother or preconditioner. The patchwise additive Schwarz blocks,

however, are able to treat these problematic modes since the basis functions in question are grouped together in this approach.

The number of CG iterations when using patchwise additive Schwarz preconditioning or elementwise additive Schwarz preconditioning on sufficiently fine meshes, increases in proportion to $1/h$. When the additive Schwarz techniques are used as smoothers in a multigrid preconditioner convergence rates independent of h are achieved. Moreover, this example shows that the use of patchwise additive Schwarz blocks can lead to convergence rates that are independent of the polynomial order, irrespective of whether they are utilized in a smoother or preconditioner, whereas the CG solvers based on elementwise blocks show an increase in the number of iterations with increasing polynomial orders.

	CG with elementwise AS preconditioner			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	60	76	135	262
3	91	81	135	262
4	170	101	140	262
5	349	135	150	262

	CG with multigrid preconditioner elementwise AS smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	12	9	9	9
3	19	12	9	9
4	35	15	11	11
5	67	19	12	12

	CG with patchwise AS preconditioner			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	43	64	109	210
3	43	64	109	210
4	43	65	109	210
5	43	64	109	210

	CG with multigrid preconditioner patchwise AS smoother (5,5)			
p \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	7	7	7	7
3	7	7	7	7
4	7	7	6	6
5	6	6	6	6

Table 5.9: Perforated plate example: Convergence behavior of a CG solver for four different preconditioners. The study considers uniform high-order finite cell meshes with varying resolutions.

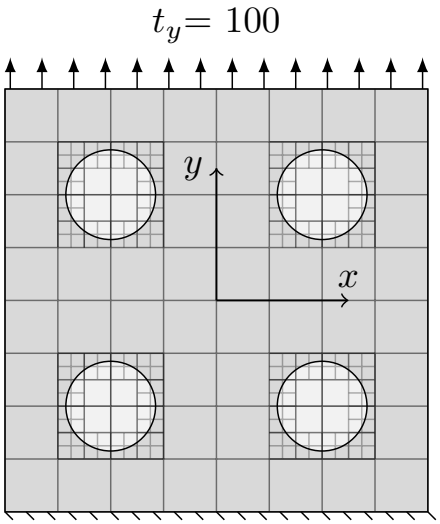


Figure 5.38: Setup of the perforated plate example involving multi-level *hp*-refinement. The mesh is refined towards the circular cavities.

Table 5.10 shows the number of iterations required by the different preconditioner and smoother configurations for multilevel *hp*-grids with quadratic elements and varying levels of refinement. When the elementwise additive Schwarz blocks are utilized as both smoothers and preconditioners, increasing the refinement depth leads to an increase in the number of iterations. The multigrid algorithm that uses these elementwise blocks for smoothing does not achieve convergence rates that are independent of the mesh parameter h. This behavior is attributed to the fact that certain small modes can remain untreated by the elementwise blocks leading to slow convergence, see Section 5.2.3. In the case of patchwise blocks, convergence rates are achieved that are independent of the refinement depth employed. Furthermore, the multigrid algorithm that employs a patchwise smoothing approach achieves mesh-independent convergence rates.

	CG with elementwise AS preconditioner			
k \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
0	60	76	135	262
1	81	220	234	1192
2	183	367	807	2114
3	294	483	1803	2566

	CG with multigrid preconditioner elementwise AS smoother (5,5)			
k \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
0	12	9	9	9
1	15	30	29	79
2	31	46	63	115
3	50	67	102	114

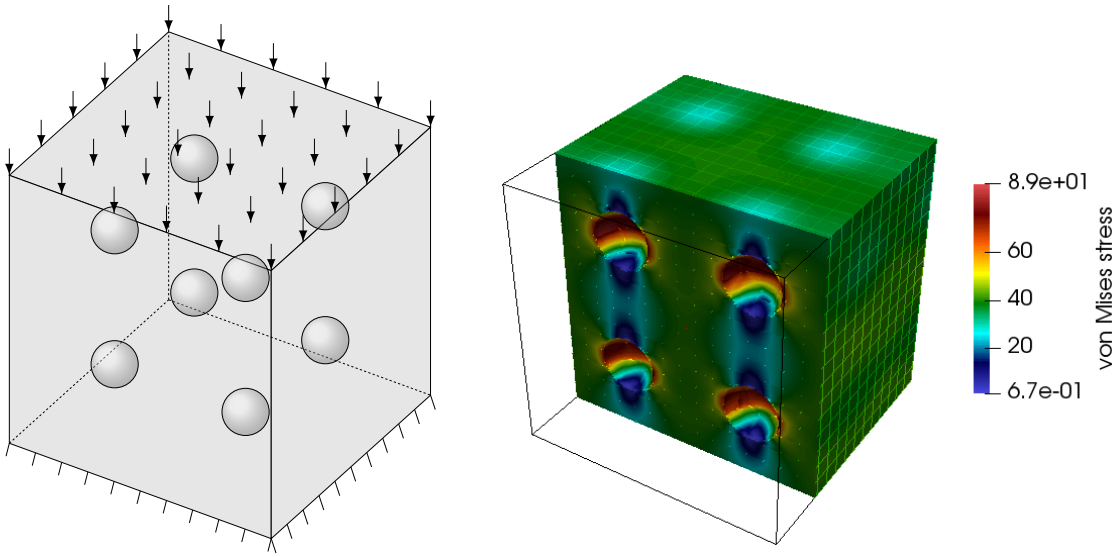
	CG with patchwise AS preconditioner			
k \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
0	43	64	109	210
1	43	64	109	210
2	43	65	109	210
3	43	64	109	210

	CG with multigrid preconditioner patchwise AS smoother (5,5)			
k \ h	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$
2	7	7	7	7
3	7	7	7	7
4	7	7	6	6
5	6	6	6	6

Table 5.10: Perforated plate example: Convergence behavior of a CG solver for four levels of refinement and four different preconditioners. The study considers multi-level *hp*-refined finite cell meshes with varying resolutions and a fixed polynomial order of $p = 2$.

5.3.4.3 Cube with spherical cavities

To assess the performance of the proposed *hp*-multigrid approach in a three-dimensional setting, we now consider a simple example consisting of a linear elastic cube of unit length with spherical cavities subject to compressional loading. The cube has the same material properties as the perforated plate in the previous example and the values of $\alpha = 10^{-8}$ and $\beta = 10^{10}$ are chosen. The radii of the cavities are chosen such that $r = 0.3\sqrt{2}$. A homogeneous pressure load $P = 100$ N/mm^2 is applied on the upper surface of the cube as shown in Figure 5.39a.



(a) Cube geometry. (b) Von Mises stress.

Figure 5.39: Cube with spherical cavities example.

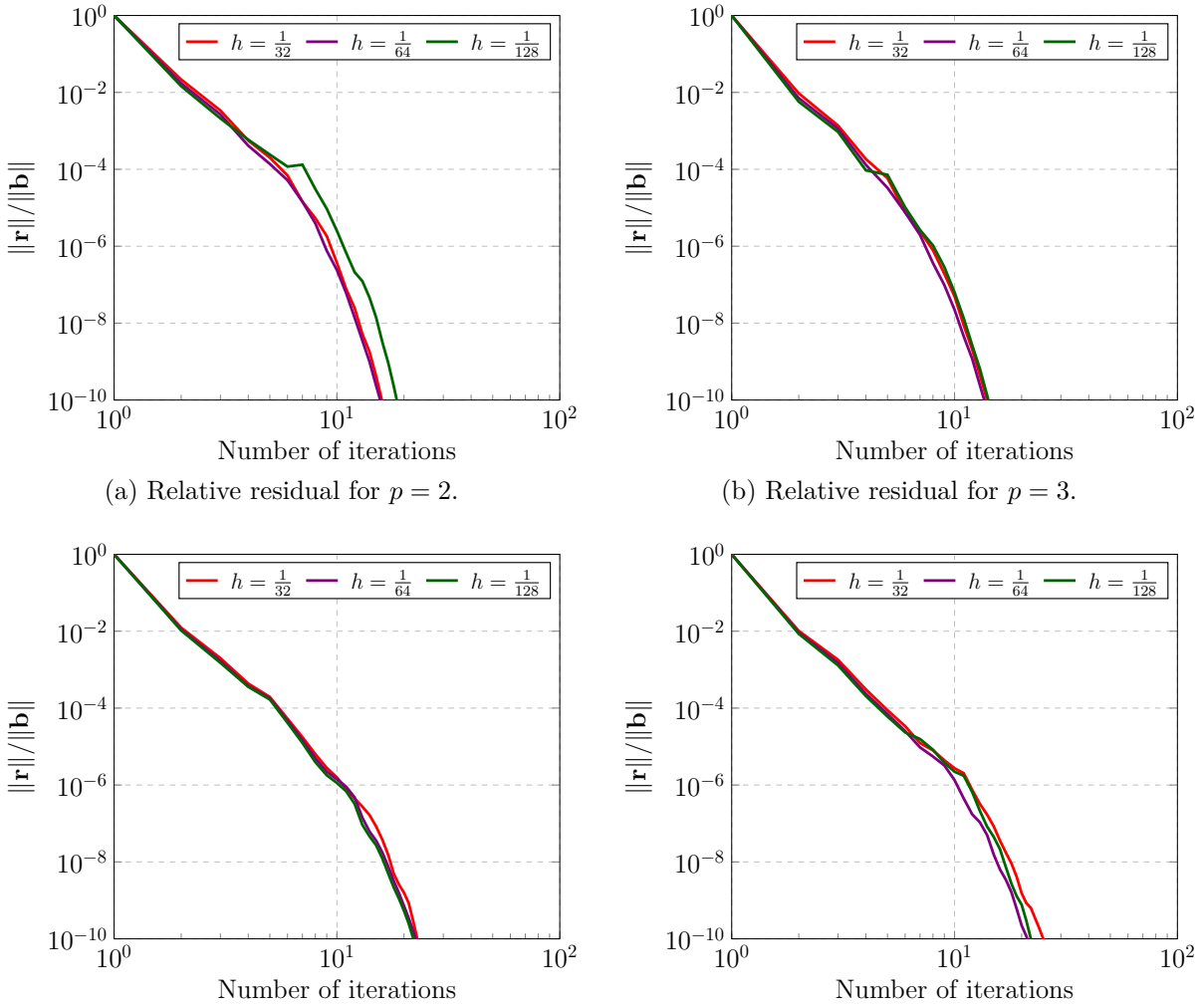
Influence of the polynomial order

The effect of the element polynomial order on the convergence of a Conjugate Gradient Solver with a p-multigrid preconditioner utilizing elementwise additive Schwarz smoothing is the subject of the following study. To this end, a sequence of meshes with varying element sizes is analyzed. The mesh size is chosen such that $h = \{\frac{1}{32}, \frac{1}{64}, \frac{1}{128}\}$ resulting in the number of unknowns summarized in Table 5.11.

p \ h	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$
2	417 339	3 188 763	24 853 851
3	727 659	5 569 083	43 448 139
4	1 337 499	10 284 507	80 451 675
5	2 246 859	17 335 035	135 864 459

Table 5.11: Number of unknowns for trunk space hexahedral elements in the study of the influence of p on the convergence of the Conjugate Gradient Solver with a p-multigrid preconditioner.

Similar to the two-dimensional benchmark cases, the proposed p-multigrid approach leads to convergence rates that are independent of the mesh size h in the example at hand as shown in Figure 5.40. The number of iterations ranges between 16 and 19 for $p = 2$, between 14 and 15 for $p = 3$, between 22 and 23 for $p = 4$ and between 22 and 26 for $p = 5$.



(a) Relative residual for $p = 2$.

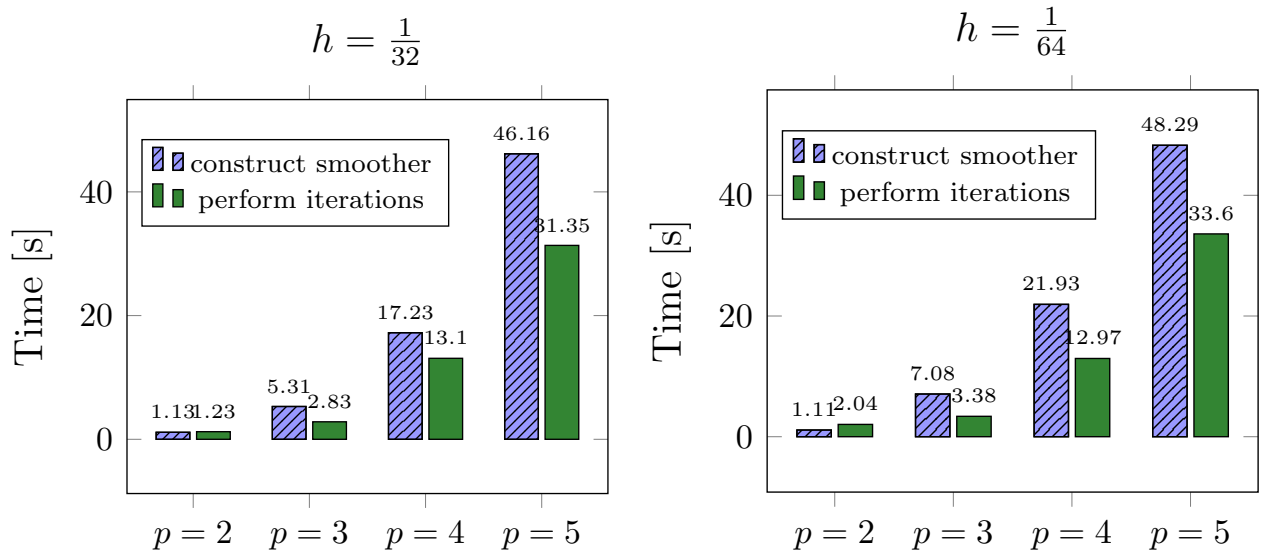
(b) Relative residual for $p = 3$.

(c) Relative residual for $p = 4$.

(d) Relative residual for $p = 5$.

Figure 5.40: Convergence of the CG solver with a p-multigrid preconditioner and elementwise additive Schwarz smoothing for hexahedral trunk space elements.

Figure 5.41 shows the computational cost of the CG solver with a p-multigrid focusing on the time spent to construct the smoother and performing the CG iterations. The latter procedure includes the application of the smoother and the solution of the coarse problem. From Figures 5.41a and 5.41b one can see that the computational cost of the solver increases for higher polynomial orders.



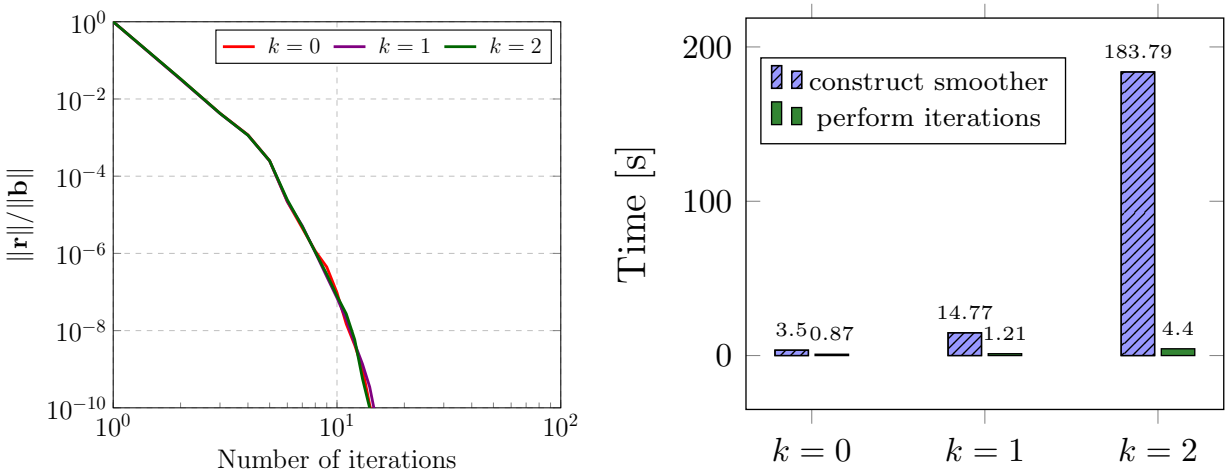
(a) CPU timings for $h = \frac{1}{32}$. All simulations are run on 96 cores.

(b) CPU timings for $h = \frac{1}{64}$. All simulations are run on 768 cores.

Figure 5.41: Execution time for the CG solver with a p-multigrid preconditioner.

Influence of the refinement level

The effect of the refinement level on the performance of the *hp*-multigrid preconditioner is now studied. Starting from an initial grid of 32^3 elements quadratic elements, the mesh is refined in two steps towards the spherical cavities yielding a total of $4.1 \cdot 10^5$, $5.2 \cdot 10^5$ and $8.9 \cdot 10^5$ DOFs for $k = 0$, $k = 1$ and $k = 2$, respectively. The patchwise additive Schwarz groups are used to construct the smoothers that are applied on every multigrid level $\ell \neq 0$ in analogy to the two-dimensional examples considered in the previous studies. This approach yields convergence rates that are independent of the refinement level as shown in Figure 5.42.



(a) Convergence behavior.

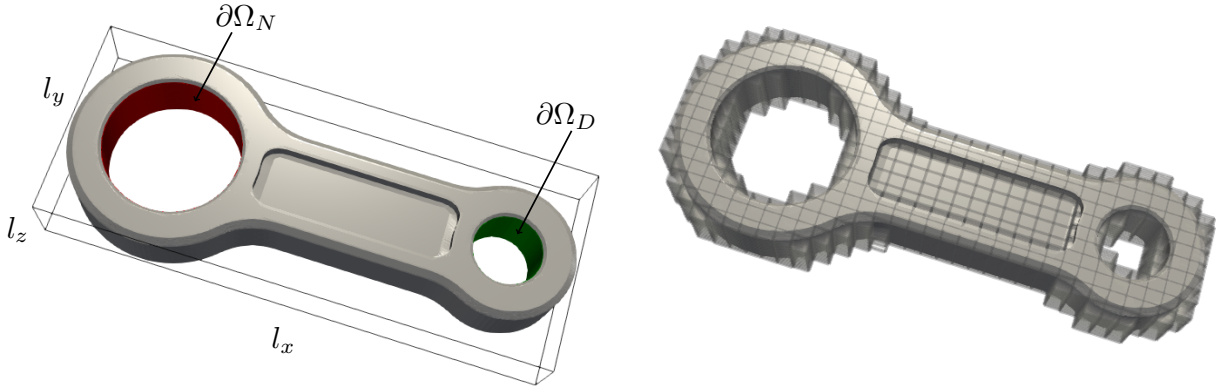
(b) Computation cost of the hierarchical multigrid solver for a set of simulations running on 384 cores.

Figure 5.42: Performance of the CG solver with an *hp*-multigrid preconditioner and patchwise additive Schwarz smoothing for a three-dimensional benchmark involving multi-level *hp*-refinement.

The computational cost of the solver for the different refinement levels considered is shown in Figure 5.42b. The results shown are obtained in hybrid simulations on 394 cores that are partitioned into 32 MPI tasks each with 6 OMP threads. As expected, the inversion of the sub-matrices for the construction of the patchwise AS smoothers is the most time-consuming operation. This procedure is performed using the sparse direct solver `Pardiso`, since it outperforms the built-in invert function of the `BOOST` library [Schling, 2011] for large matrices. The time spent setting up the patchwise smoothers can be further reduced by the use of a more optimized inversion algorithm. Moreover, it is possible to conceive different strategies for the grouping of basis functions that yield smaller group sizes and therefore lower computation times.

5.3.4.4 Loading of an aluminum rod

The next three-dimensional example considers the loading of an aluminum rod. The rod has an elastic modulus E = 70 GPa and a Poisson's ratio $\nu = 0.3$. Connecting rods are typically used in mechanical engineering to transform the linear movements of a piston into circular motion of a crankshaft in combustion engines.



(a) Rod geometry and boundary condition surfaces.

(b) Finite cell mesh.

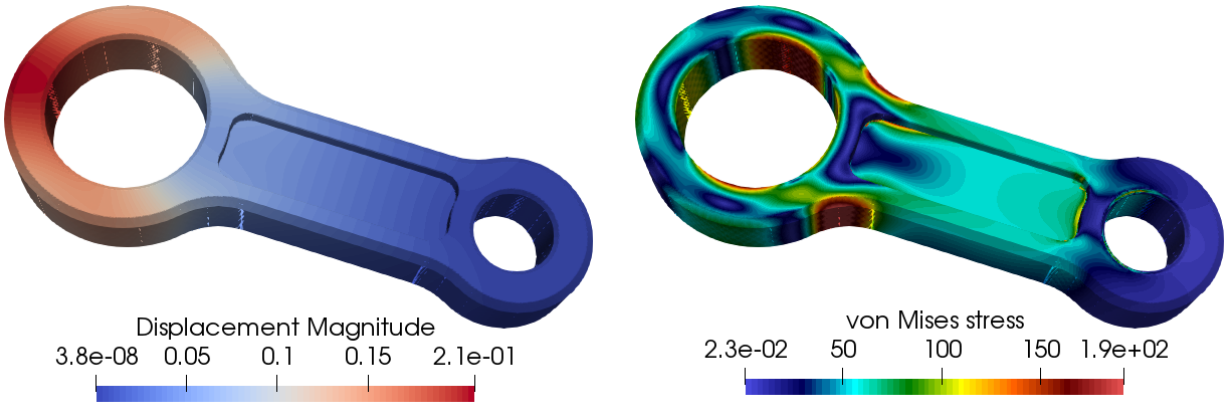
Figure 5.43: Loading of an aluminum connecting rod.

Figure 5.43a shows the geometry of the connecting rod with $l_x = 150$ mm, $l_y = 60$ mm and $l_z = 20$ mm. The cylindrical surfaces labeled Ω_D are fixed using the penalty method with a penalty parameter $\beta = 10^6$. A surface load $t_x = 10\,\mathrm{N/mm^2}$ is applied on the cylindrical surfaces labeled Ω_N in the direction of the rod's shaft. The resultant force acting on the rod is $F_x = 2\pi rh \cdot t_x = 2\pi \cdot 20 \cdot 10 \approx 12.566\,\mathrm{kN}$.

In this example, the performance of the multigrid solution techniques developed in this thesis is assessed in a practical engineering application. To this end, three finite cell discretizations of the rod with $h \in \{2, 1, 0.5\}$ are considered and the polynomial degree of the grids chosen as $p \in \{2, 4\}$. Table 5.12 summarizes the number of DOFs in each grid. An octree scheme with a depth of 3 is used for the integration of the element matrices. The resulting linear systems are solved using a preconditioned CG solver. In the study at hand, the convergence behavior and execution time of the iterative solver with elementwise additive Schwarz preconditioner and the hierarchical multigrid preconditioner are compared in a practical engineering application. Figure 5.44 shows the displacement magnitude and von Mises stress in the aluminum rod that result from the loading process.

		DOFs		
h	mesh resolution	$p = 2$	$p = 3$	$p = 4$
2	$75 \times 30 \times 10$	156 594	271 596	492 285
1	$150 \times 60 \times 20$	1 066 059	1 861 425	3 410 673
0.5	$300 \times 120 \times 40$	7 880 868	13 755 963	25 365 804

Table 5.12: Summary of the number of DOFs for the different discretizations considered in the aluminum rod example.



(a) Displacement magnitude.

(b) Von Mises stress.

Figure 5.44: Visualization of the displacement magnitude and the von Mises stress.

The convergence behavior for the different finite cell meshes considered in this example is shown in Figure 5.45. Both preconditioners yield the expected convergence rates i.e. the elementwise additive Schwarz preconditioner yields convergence rates that are proportional to h^{-1}, while the use of the p-multigrid preconditioner leads to convergence rates that are independent of the mesh size. Moreover, there is only a minimal difference in the convergence behavior of different polynomial orders for all preconditioners.

Although multigrid solution techniques lead to lower iteration counts than single-grid methods, they do not necessarily lead to faster execution times than single-grid solvers. To study the computational efficiency of the single-grid and multigrid preconditioning techniques suggested in this thesis, an analysis is conducted in which the number of processors is increased in proportion to h^{-1} from 48 to 3072 cores (1 to 64 nodes). The amount of time needed to perform the CG iterations is monitored for the different meshes and preconditioner configurations. All simulations runs are performed on the SuperMUC-NG system at the Leibniz Supercomputing Center and comprise of hybrid computations with 8 MPI tasks per node and 6 OpenMP threads per task.

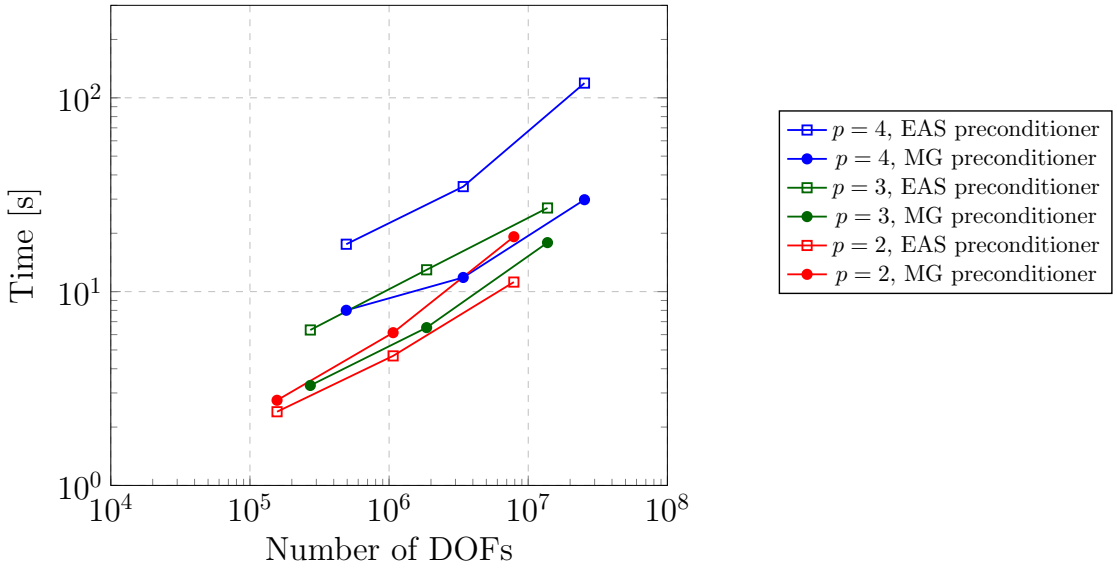
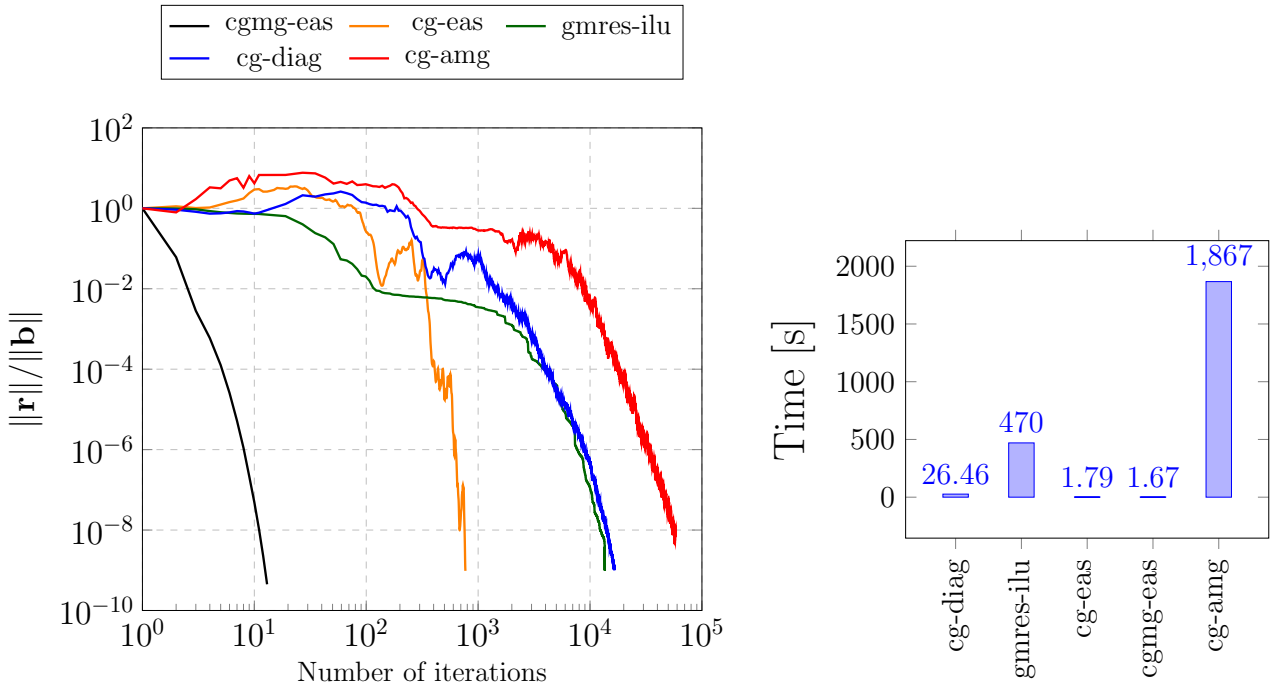


Figure 5.46: Comparison of the execution time of a CG solver for two different preconditioners: an elementwise additive Schwarz preconditioner (EAS) and a p-multigrid preconditioner (MG).

Figure 5.46 shows the execution time of the CG solver for the two different preconditioners considered in this study. The CG solver using the multigrid preconditioner outperforms the single-grid solver for all polynomial orders higher than $p \geq 3$. The example at hand clearly shows that multigrid preconditioning can help improve the efficiency of high-order immersed FE-analysis. The multigrid preconditioned CG solver presented in this thesis can, however, be further optimized to improve its weak scalability. In the current study, the solution of the coarse problem with $p = 1$ is the most time-consuming routine in the multigrid solver. Improving the solution of the coarse system will have a positive effect on the overall efficiency of the proposed solver.

Comparison to different iterative solvers

The final numerical study in this example compares the preconditioning techniques proposed in this thesis, i.e. additive Schwarz preconditioning and an *hp*-multigrid scheme for immersed methods, to different solvers and preconditioners available in the `AztecOO` package of `Trilinos`. To this end, the linear system arising from the aluminum rod with $h = 2$ and $p = 3$ is solved using different iterative solvers. All simulations are run on four nodes of the SuperMUC-MG and each solver's convergence behavior and execution time are monitored.

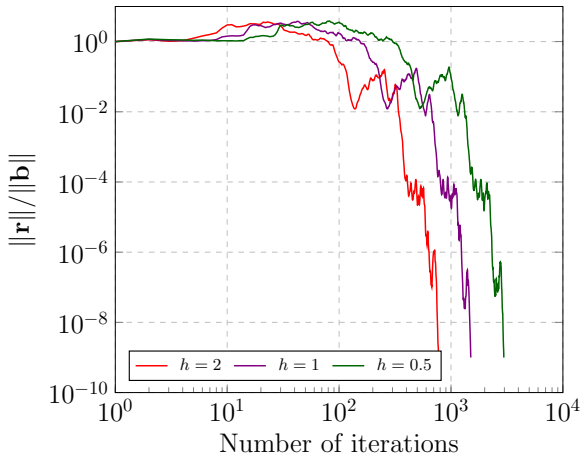


(a) Relative residual.

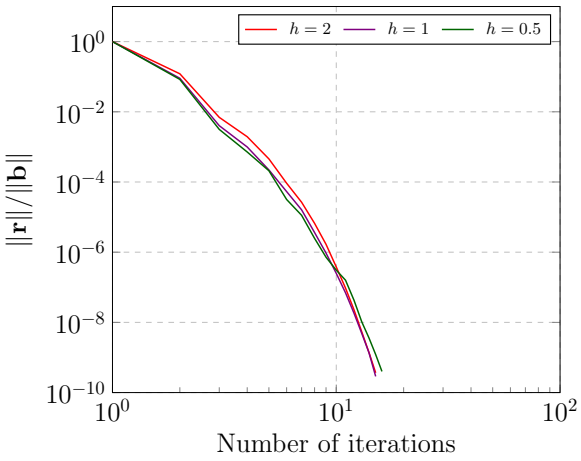
(b) Solver time.

Figure 5.47: Comparison of the convergence behavior and execution time of different iterative solvers in the aluminum rod example.

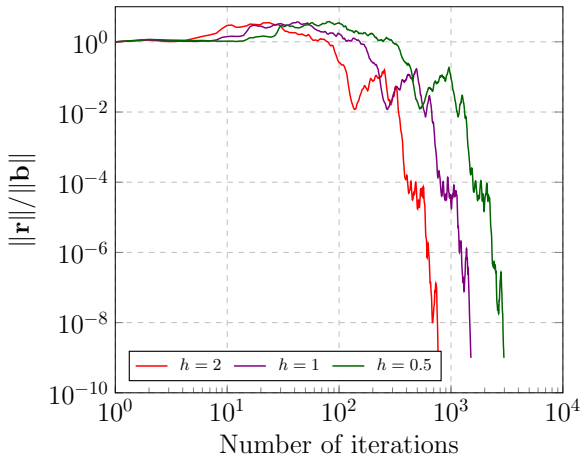
In the study at hand, the convergence behavior and execution time of five different iterative solution techniques is compared in Figures 5.47a and 5.47b. The solvers applied include: *i)* a CG solver with diagonal scaling, denoted by cg-diag, *ii)* a CG solver with elementwise additive Schwarz preconditioning, denoted by cg-eas, *iii)* a CG solver with a *p*-multigrid preconditioner and elementwise additive Schwarz smoothing, denoted by cgmg-eas, *iv)* a CG solver with an algebraic multigrid solver based on smoothed aggregation, denoted by cg-amg and *iv)* a GMRES solver with an incomplete LU preconditioner, gmres-ilu. Note that the solvers cg-diag, cg-amg and grmes-ilu are available in the `AztecOO` package in `Trilinos`. The results in Figure 5.47 show that the additive Schwarz-based solution techniques put forward are well suited for immersed systems as they not only show superior convergence behavior than conventional solvers but also exhibit lower computational times.



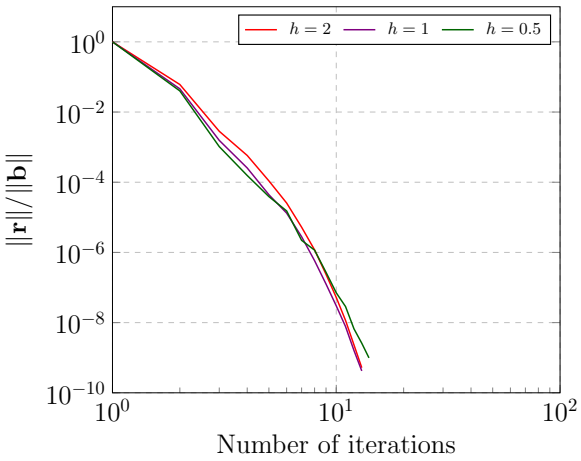
(a) Additive Schwarz preconditioner for $p = 2$ meshes.



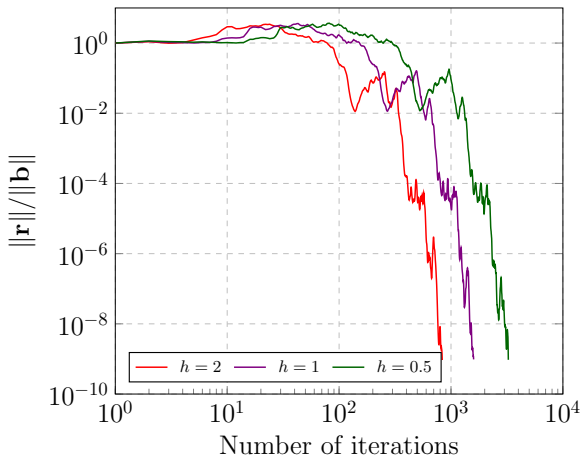
(b) p-multigrid preconditioner for $p = 2$ meshes.



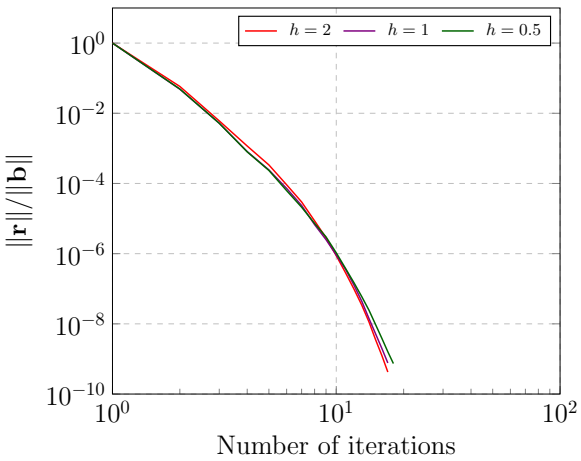
(c) Additive Schwarz preconditioner for $p = 3$ meshes.



(d) p-multigrid preconditioner for $p = 3$ meshes.



(e) Additive Schwarz preconditioner for $p = 4$ meshes.



(f) p-multigrid preconditioner for $p = 4$ meshes.

Figure 5.45: Aluminium rod example: Convergence of a CG solver with two different preconditioners; *i)* an elementwise additive Schwarz preconditioner and *ii)* a p-multigrid preconditioner and elementwise additive Schwarz smoothing. The considered grids comprise of trunk space hexahedral elements with different polynomial orders.

Chapter 6

Application of the finite cell method to metal additive manufacturing

Numerical simulations are becoming increasingly important in the field of metal additive manufacturing (AM). In general, simulations in the field of metal additive manufacturing can be divided into two groups; AM process simulations and AM product simulations. Process simulations involve a model-based representation of the printing procedure and aim to reproduce the relevant physical effects and changes taking place using numerical methods. Such simulations provide valuable insight into the process' underlying physical phenomena and can be used to improve process parameters leading to better quality products. Product simulations are commonly performed to investigate the suitability of the final AM product for its intended use i.e. whether the product exhibits the desired physical properties e.g. mechanical response, porosity, thermal conductivity. Process and product simulations can be performed in tandem to optimize the properties of the final product.

The chapter at hand portrays how the finite cell method, parallelism and multi-level *hp*-refinement can be applied in the context of metal additive manufacturing. First, it is shown how the developed methods can be used in product simulations of additively manufactured micro-architectured components. Starting from CT scans of the specimens, its is shown how the mechanical properties of bodies with complex microstructures can be determined. The latter part of this chapter is dedicated to the simulation of the selective laser melting process (SLM) using the methods presented in this thesis. The ease of mesh generation in FCM is advantageous in these simulations, as it allows the consideration of bodies with complex geometrical shapes. Moreover, the growth of the additive manufacturing part can be easily modeled via an evolving embedded domain Ω_{phys}. Thermal- and thermo-mechanical simulations are presented that show the benefit of using FCM, parallelism and multi-level *hp*-refinement for AM process simulations.

6.1 Virtual material characterization of AM products [1]

The physical properties of additively manufactured products highly depend on the parameters of the process used to fabricate them. It is common for these products to exhibit

[1] The following section is based on [Korshunova et al., 2020]

significant variations from the intended geometry such as process-induced porosity, different surface roughness, or other undesired morphological variations that affect the product's overall physical behavior. Finite element simulations provide a means of assessing the usability of these imperfect products through virtual material characterization. These simulations mainly take computer tomographic images of the products as their input. Boundary-conforming mesh generation for imperfect AM geometries is quite involving especially when small geometrical features need to be resolved. The finite cell method is better suited from image-based FE-simulations than boundary-fitting methods since it can be readily applied to the CT scans of the AM products. The following section presents how the finite cell method and parallel computing techniques can be applied in the material characterization of additively manufactured porous micro-architectured components. The macroscopic behavior of such components is significantly influenced by morphological imperfections that can be easily captured using FCM.

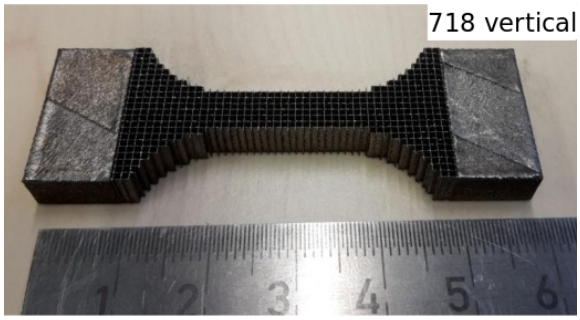
The results in this section are based on the work presented in [Korshunova et al., 2020]. This publication considers three different numerical methods for the characterization of imperfect AM products based on CT scans namely *i)* Direct numerical simulations (DNS) using high-order Voxel FEM *ii)* Direction numerical simulations using the finite cell method and *iii)* An approach that combines the finite cell method and numerical homogenization techniques. The modularity and generality of the finite cell method and the parallel approach presented in Section 4.4 allows these three methods to be incorporated into `AdhoC++`. Only the first two approaches will be considered in the following section for the sake of brevity.

6.1.1 Characterization of a microporous metallic structure

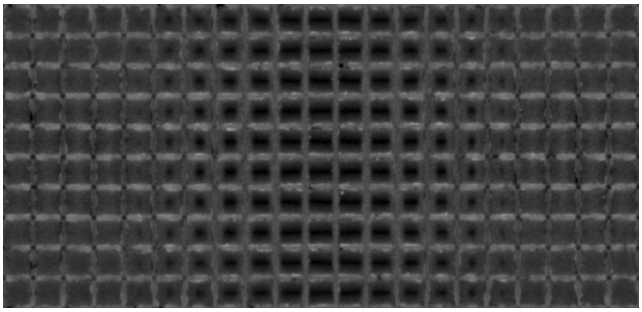
The following study seeks to determine the macroscopic behavior of additively manufactured foam structures produced by Selective Laser Melting. The structures are fabricated by Siemens AG and are made up of an Inconel®718 alloy with a Young's modulus $E = 190\text{GPa}$ and a Poisson's ratio $\nu = 0.294$. Tensile experiments are performed on three specimens of the foam structure yielding the macroscopic properties provided in Table 6.1. The wide range of values is attributed to the difficulty in controlling the SLM process parameters at the mircoscopic level resulting in manufacturing defects that affect the reproducibility of the parts. The specimens are of type specimen 600 with a geometrical form as shown in Figure 6.1a.

Specimens	E_{zz} [MPa]	ν_{xy} [−]	ν_{xz} [−]
600 L1-L3	15 339. . . 26 731	-0.04. . . 0.13	0.05. . . 0.14
600 L2 Narrow	20 851	-0.04	0.08
600 L2 wide	25 915	-	-

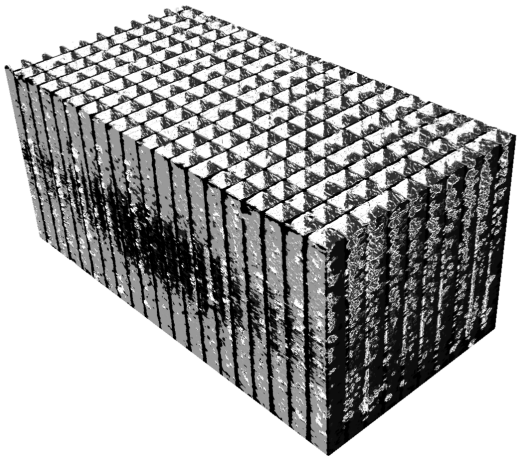
Table 6.1: Specimen 600: Results of the tensile experiments of three specimens.



(a) Inconel®718 specimen 600.



(b) Coronal slice of the unsegmented CT scan.



(c) Voxel model after segmentation - Specimen 600 L2.

Figure 6.1: Geometry of the Inconel®718 specimen 600

In order to carry out the virtual testing through numerical simulations, one of the Specimen 600 L2 foams was subjected to computer tomography and segmented using deep learning techniques as outlined in [Korshunova et al., 2020]. Figure 6.1b shows a slice through the unsegmented CT scan while Figure 6.1c shows the final 3D voxel model after segmentation. The segmented CT scan is used as input of the numerical simulations and has a resolution of $800 \times 368 \times 400$.

The simulation pipeline for the DNS with voxel FEM and the finite cell method are rather similar with the major difference being the setup and nature of the computational mesh. Voxel FEM fully resolves the geometry of the specimen by taking one element per voxel while the immersed approach in FCM allows the use of much coarser meshes with several voxels within a single element and resolves the specimen's geometry only at integration level. Standard Gauss quadrature is applied in voxel FEM simulations while the voxel integration approach presented in Section 2.3.2.1 is used in all FCM computations. The virtual tensile tests are performed by applying a virtual displacement field on the top and bottom end surfaces of the foam specimen in x-direction. To guarantee comparability of the results, the penalty method is used in all simulations for the application of the boundary conditions. Furthermore, elements from the trunk space are used in both the voxel FEM and FCM simulations. Two different discretizations are used for the FCM simulations, the first mesh consists of a $100 \times 46 \times 100$

elements, with each element containing $8 \times 8 \times 4$ voxels, while the second FCM mesh comprises $200 \times 92 \times 200$ elements, with each element containing $4 \times 4 \times 2$ voxels.

The effective elastic modulus of the foam specimen is determined from the quotient of the average stress and average strain obtained by performing linear-elastic simulations using the described setup. The parallelization scheme using fully distributed mesh data structures is utilized to this end. Furthermore, the elementwise additive Schwarz preconditioning approach is used in conjunction with a parallel Conjugate Gradient solver for the solution of the linear system. A convergence study of the elastic modulus is performed by raising the polynomial order of the meshes such that $p \in [1, 6]$ for the FCM simulations and $p \in [1, 4]$ for the voxel FEM computations. The total number of unknowns in each computation is summarized in Table 6.1.

	FCM		voxel FEM
p	$100 \times 46 \times 100$ elements	$200 \times 92 \times 200$ elements	$800 \times 368 \times 400$ elements
1	1 067 010	3 421 977	118 600 659
2	3 953 580	12 714 699	451 734 033
3	6 840 150	22 007 421	784 867 407
4	12 272 916	39 560 592	1 428 159 774
5	20 251 878	65 374 212	-
6	31 504 386	101 839 776	-

Table 6.2: Number of DOFs in the different simulation setups.

Convergence of the Young's modulus E_{xx}

The Young's modulus in x-direction is determined in each numerical simulation and compared to the solution of the high-order voxel FEM simulation with a polynomial order of $p = 4$. This value of $E_{xx} = 23584.62\,\text{MPa}$ is close to the average value obtained from the experiments of the foam specimens of type 600 L2, see Figure 6.1. It should be noted, however, that the quality of the overkill solution obtained using high-order voxel FEM is subject to modeling errors due to the stepwise definition of the elastic modulus and dependent on the resolution of the CT scan. The relative error in E_{xx} with respect to the overkill solution is plotted over the number of degrees of freedom and displayed in Figure 6.2. The results of this convergence study show that the finite cell method is able to provide acceptable values of the elastic modulus without the need for fully resolving the microstructure with the computational mesh. The coarsest FCM discretization provides an error of 9.74% with respect to the overkill solution while the finest one provides an error of 5%. Voxel FEM, as expected, provides a better approximation of the elastic modulus since the mircostructure is fully resolved by the mesh. This high resolution, however, comes at a cost since more computational resources are needed to run these simulations. The simulation with 1.42 billion DOFs, for example, required a minimum of 700 nodes of the SuperMUC-NG cluster. The study at hand also shows the benefit of using higher order polynomials as they lead to better quality approximations of E_{xx} in both voxel FEM and FCM.

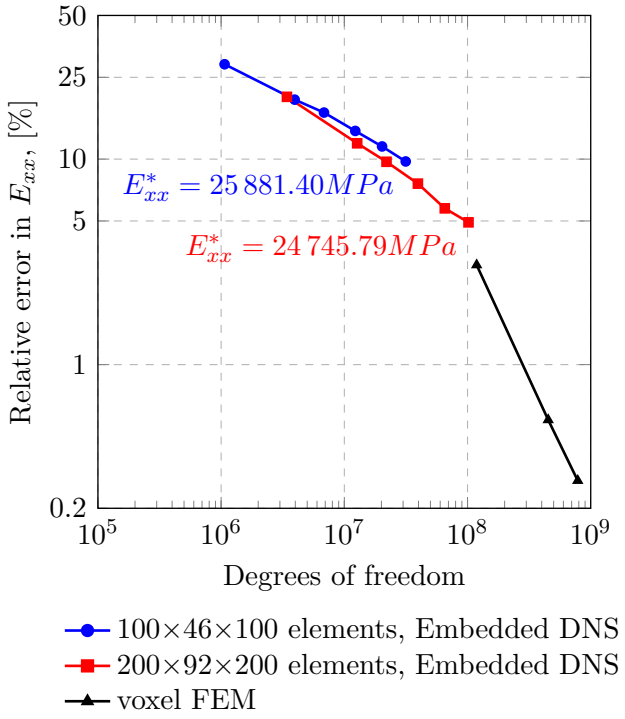


Figure 6.2: Specimen 600: Convergence of the directional Young's modulus E_{xx}.

6.2 Modeling heat transfer in selective laser melting

Selective laser melting is a laser powder bed fusion (LPBF) process in which a concentrated laser beam is used to melt selected regions of a powder bed, allowing a part to built up in a layer-by-layer fashion. Different modeling approaches can be used when simulating the SLM process. These approaches consider different spatial and temporal scales and can be grouped into powder models, microstructure models and part-scale models [King et al., 2015]. In this thesis, only part-scale models are considered for the thermal simulation of SLM processes: a high-fidelity approach that explicitly resolves the laser spot and an m-layer-by-m-layer model that averages the heat input during a heating step. The section at hand shows how the finite cell method can be utilized in the simulation of the SLM process. It begins with a summary of the governing equations before illustrating how FCM can be used in high-fidelity and part-scale layerwise simulations.

6.2.1 Governing equations

The heat transfer during the SLM process can be modeled using the balance of energy equation that relates the rate of change in volumetric enthalpy H to the heat input per unit volume Q and the heat conduction flux $\mathbf{q}$ as shown in (6.1), where the time is denoted by the variable t and the conductivity by κ. Note that the effects of mass transfer are neglected in the below approach.

$$\frac{\partial H}{\partial t} - \nabla \cdot \mathbf{q} = \frac{\partial H}{\partial t} - \nabla \cdot (\kappa \nabla T) = Q. \tag{6.1}$$

The volumetric enthalpy H is defined as a function of the temperature T, a reference temperature T_{ref} and a set temperature-dependent quantities, the specific heat capacity c, density ρ, latent heat L and a phase change function f_{pc}, as shown in (6.2).

$$H(T) = \int_{T_{ref}}^{T} \rho c \, \mathrm{d}T + \rho L f_{pc}. \tag{6.2}$$

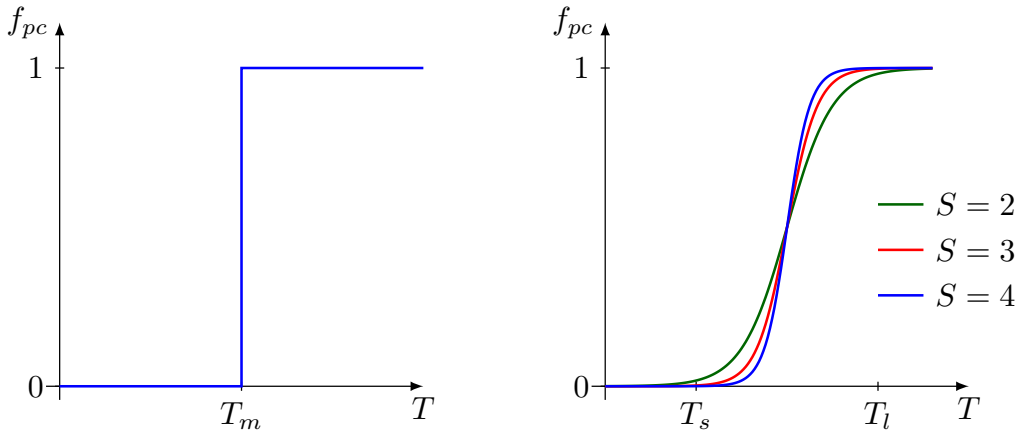
The phase change function f_{pc} describes the relative volume fraction of the solid and liquid phases of a material as a function of the temperature. Its shape depends on the nature of the SLM process and the material's thermal properties. Pure materials generally undergo isothermal phase change from the solidus to liquidus states (and vice versa) and are characterized by a distinct melting (solidification) temperature T_m. In this case, a Heavyside step function can be used to described f_{pc} as shown in the below equation.

$$f_{pc}(T) = \begin{cases} 1 & T \leq T_m \\ 0 & T > T_m \end{cases}. \tag{6.3}$$

Most alloys do not have a distinct melting temperature but rather solidify over a temperature range. For numerical reasons, it is common to use a smooth regularization of f_{pc} for both pure materials and alloys given by

$$f_{pc}(T) = \frac{1}{2}\left[\tanh\left(S\frac{2}{T_l - T_s}\left(T - \frac{T_s + T_l}{2}\right)\right) + 1\right]. \tag{6.4}$$

The shape and slope of the regularized phase change function is controlled by a parameter S and the solidification range defined by the temperature T_s at which the material is completely solid and the temperature T_l at which the material is completely molten. Figure 6.3 illustrates the phase change functions for the isothermal and non-isothermal cases.



(a) Isothermal phase change. (b) Non-isothermal phase change.

Figure 6.3: Phase change functions for the isothermal and non-isothermal cases.

Inserting (6.2) in (6.1) yields the strong form of the nonlinear heat equation with phase change that reads: Find T for all $\boldsymbol{x} \in \Omega$ that satisfies the equation

$$\rho c \frac{\partial T}{\partial t} + \rho L \frac{\partial f_{pc}}{\partial t} - \nabla \cdot (\kappa \nabla T) = Q, \tag{6.5}$$

subjected to an initial condition and the Dirichlet and Neumann boundary conditions in (6.6) with the term T_0 representing the initial temperature and $\bar{T}$ denoting a prescribed temperature. It is common to use a prescribed temperature to model the heat conduction through the building platform e.g. [Chiumenti et al., 2017].

$$\text{Initial condition:} \quad T(\boldsymbol{x}, t=0) = T_0 \text{ in } \Omega, \tag{6.6a}$$

$$\text{Prescribed temperature :} \quad T(\boldsymbol{x}, t) = \bar{T} \text{ on } \partial\Omega_D, \tag{6.6b}$$

$$\text{Heat loss over the boundaries :} \quad (\kappa \nabla T) \cdot \mathbf{n} = q \text{ on } \partial\Omega_N. \tag{6.6c}$$

In SLM, the total heat loss q over the boundary $\partial\Omega_N$ is driven by three main mechanisms, namely, *i)* the convection of heat through the surrounding environment that is expressed using Newton's law. It takes into account the ambient temperature T_e and the heat transfer coefficient for convection between the printed material and the environment in the printing chamber as shown in (6.7a), *ii)* the conduction of heat through the powder that is also computed using Newton's law as shown in (6.7b) and takes into account the average temperature of the powder T_p in areas away for the HAZ and the heat transfer coefficient by conduction h_{cond} between the printed material and the powder and *iii)* heat radiation that is computed using the Stefan-Boltzmanns's law as shown in (6.7c). Here, the Stefan-Bolzmann constant σ and the emissivity of the radiating surface ϵ are related to the temperature T of the radiating surface and the ambient temperature T_e.

$$\text{Convection through the surrounding :} \quad q_{conv}(T) = h_{conv}(T_e - T) \text{ on } \partial\Omega_e, \tag{6.7a}$$

$$\text{Conduction through the powder :} \quad q_{cond}(T) = h_{cond}(T_p - T) \text{ on } \partial\Omega_{cond}, \tag{6.7b}$$

$$\text{Heat loss by radiation :} \quad q_{rad}(T) = \sigma\epsilon(T_e^4 - T^4) \text{ on } \partial\Omega_e. \tag{6.7c}$$

Following [Chiumenti et al., 2017] it is possible to express the heat loss by radiation in (6.7c) in terms of a Newton's law as

$$q_{rad}(T) = h_{rad}(T_e - T) \text{ on } \partial\Omega_e, \quad \text{with} \quad h_{rad} = \sigma\epsilon\left(T^3 + T^2 T_e + T T_e^2 + T_e^3\right). \tag{6.8}$$

We follow the approach in [Chiumenti et al., 2017] and utilize the effective heat loss term q_{loss} presented in (6.9) that combines the contributions of convection and radiation since it is hard to distinguish the effect of these heat transfer modes in practice. It should be noted that the resulting heat transfer coefficient h_{loss} has no direct physical interpretations and has to be calibrated with experiments.

$$q_{loss}(T) = h_{loss}(T_e - T) \text{ on } \partial\Omega_e. \tag{6.9}$$

6.2.2 Spatial and temporal discretization

The finite element method can be readily applied to the strong form in (6.5) resulting in continuous the weak form written as

$$\int_\Omega \psi \rho c \frac{\partial T}{\partial t} \, \mathrm{d}\Omega + \int_\Omega \psi \rho L \frac{\partial f_{pc}}{\partial t} \, \mathrm{d}\Omega + \int_\Omega \nabla\psi \cdot (\kappa \nabla T) \, \mathrm{d}\Omega = \int_\Omega \psi \, Q \, \mathrm{d}\Omega + \int_{\partial\Omega_e} \psi \, q_{loss} \, \mathrm{d}\Omega + \int_{\partial\Omega_{cond}} \psi \, q_{cond} \, \mathrm{d}\Omega, \tag{6.10}$$

where ψ is a test function that vanishes on $\partial\Omega_D$. This equation can be discretized within the Bubnov-Galerkin framework yielding the semi-discrete system of equations given by

$$\mathbf{C}\dot{\mathbf{T}} + \dot{\mathbf{L}} + \mathbf{K}\mathbf{T} = \mathbf{F}, \tag{6.11}$$

with the capacitance matrix $\mathbf{C}$, conductivity matrix $\mathbf{K}$, load vector $\mathbf{F}$, latent heat vector $\mathbf{L}$ and the vector of unknown temperature coefficients $\mathbf{T}$. The underlying integrals for the computation of the terms in (6.11) and the details pertaining to the temporal discretization of (6.11) can be found in [Kollmannsberger et al., 2018].

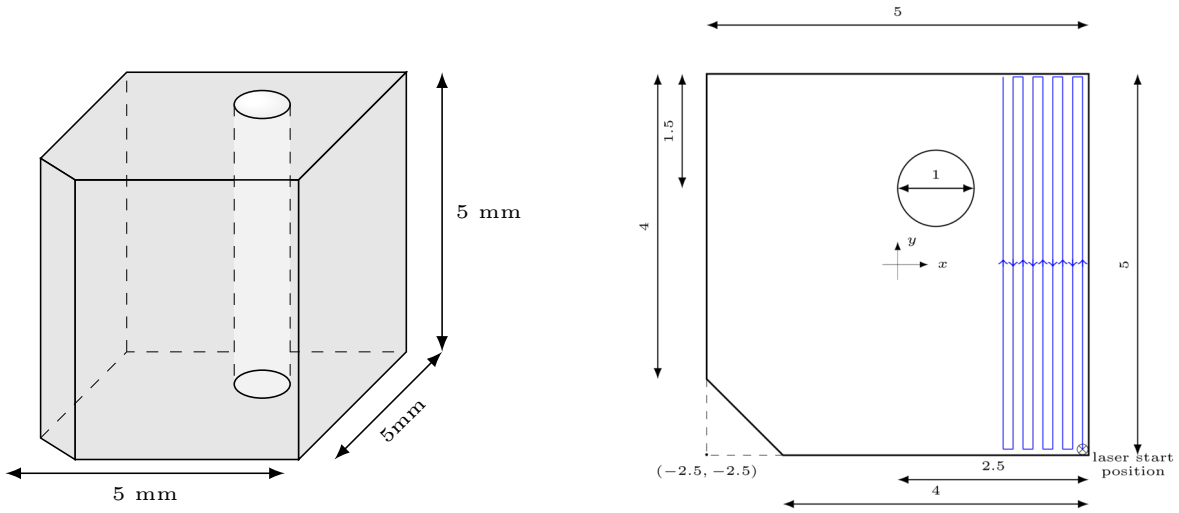
6.2.3 The heat source in high-fidelity SLM simulations

In high-fidelity thermal SLM simulations, the heat input due to the laser is modeled by incorporating the shape and size of the laser spot in the numerical model. In this thesis, the strategy presented in [Özcan et al., 2019] is used and the heat input modeled as a moving body load that is based on Goldak's ellipsoidal model [Goldak et al., 1984]. Equation 6.12 shows the heat source $q(x, y, z)$. The variables x_c, y_c, z_c represent the center of the laser spot while l_x, l_y and l_z denote the semi-axes of the ellipsoid and Q denotes the laser power.

$$q(x,y,z) = \frac{6\sqrt{3}\,Q}{\pi\sqrt{\pi}\,l_r\,l_r\,l_d}\, e^{-3\left(\frac{x-x_c}{l_r}\right)^2} e^{-3\left(\frac{y-y_c}{l_r}\right)^2} e^{-3\left(\frac{z-z_c}{l_d}\right)^2}, \tag{6.12}$$

6.2.3.1 Simulating the fabrication on an aluminum specimen

The following benchmark example shows the interplay of the finite cell method, multi-level *hp*-refinement and parallel computing for high-fidelity thermal SLM simulations. It considers an aluminum specimen with a simple geometry, see Figure 6.4a, that is made up of an aluminum AlSi10Mg alloy. Figure 6.5 shows the material properties of the alloy that are relevant for the thermal analysis. The emissivity of the metal is set to 0.47 and a latent heat value of 270 000 J/Kg is used while the initial temperature is set to 25°C.



(a) Final geometry of the aluminum specimen.

(b) Visualization of the laser path.

Figure 6.4: Geometry of the aluminum specimen and visualization of the laser path.

Simulation parameters

Apart from the material properties given in Figure 6.4, it is necessary to provide additional process parameters when performing high-fidelity simulations such as the laser specifications. In the example at hand, the following process parameters are used.

Laser power	350 W
Laser velocity	1100 mm/s
Laser spot radius	0.04 mm
Hatch distance	0.08 mm

Table 6.3: Process parameters

Problem setup

The example at hand is intended to verify the parallel implementation and the data transfer algorithms presented in Section 4.4.3 and therefore uses certain simplifications that aim at reducing the computational cost. First, it is assumed that the specimen has already been built to a height of 1 mm and the simulation commences at this height. Secondly, the base plate is not modeled since only the evolution of the melt-pool is of interest in the current study.

The specimen geometry is discretized with a mesh comprising of $32 \times 32 \times 16$ elements. This mesh is refined in four steps towards the laser spot using the multi-level *hp*-method. Since the laser position is known a priori, it is possible to pre-refine the mesh along the laser track and update the refinement every t_r time steps. This increases the efficiency of the simulation as load balancing only needs to be performed every t_r time steps when the discretization changes.

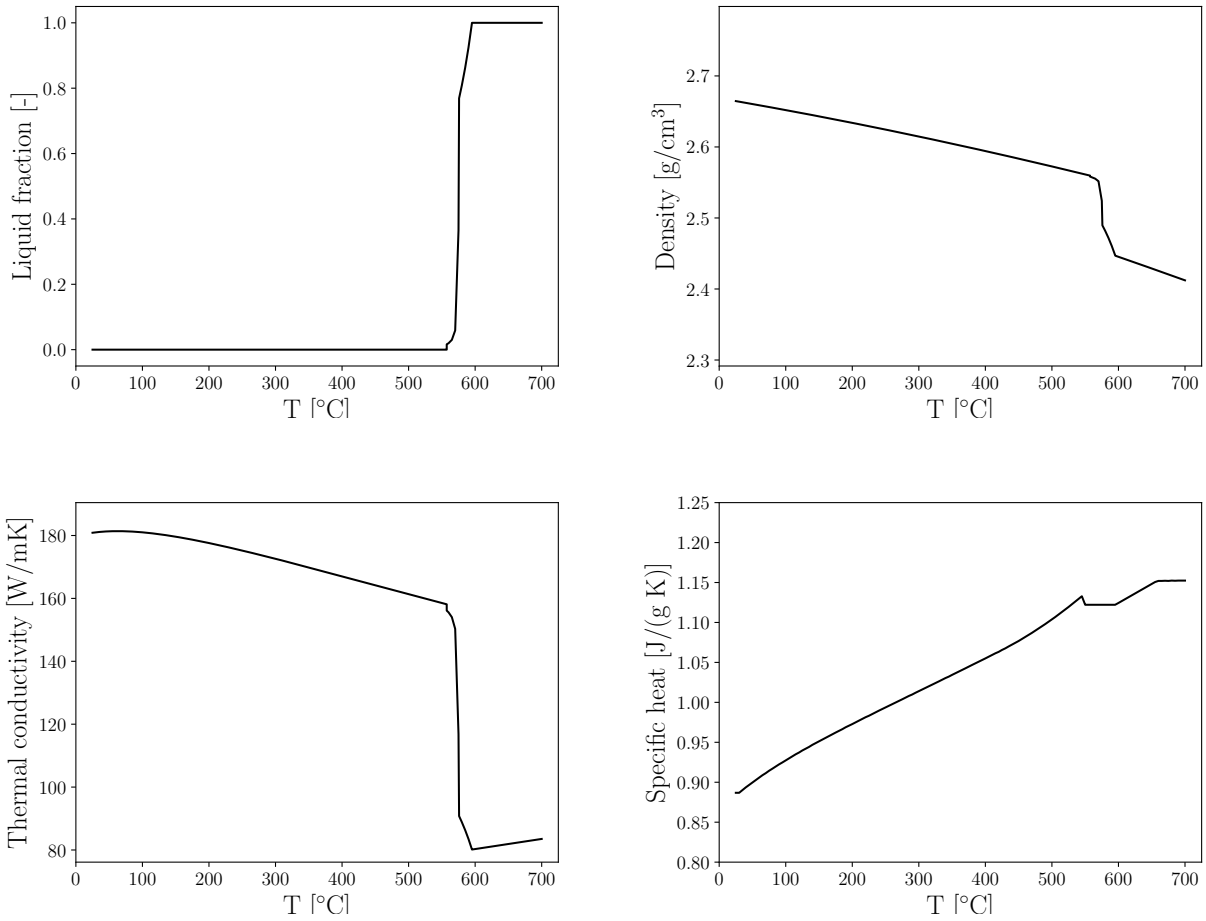
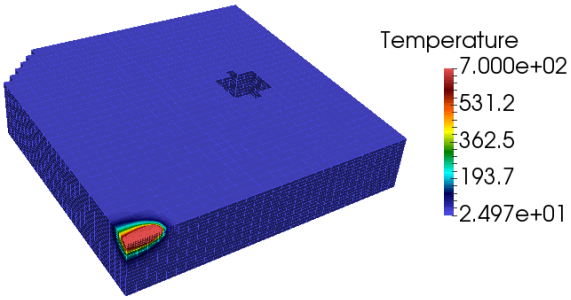


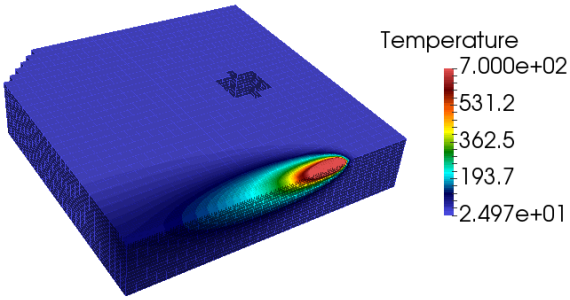
Figure 6.5: Material properties of the Aluminium AlSi10Mg alloy.

To reduce the pressure on the file system and the size of output files, one can prescribe the time steps that should be post-processed. In the example at hand, the performance of our framework for two different polynomial orders is compared, i.e. $p = \{1, 2\}$. A total of 200 time steps are computed, in which the laser travels a distance of 8 mm. The refinement is updated every $t_r = 10$ time steps while the temperature field is post-processed every $t_p = 20$ time steps.

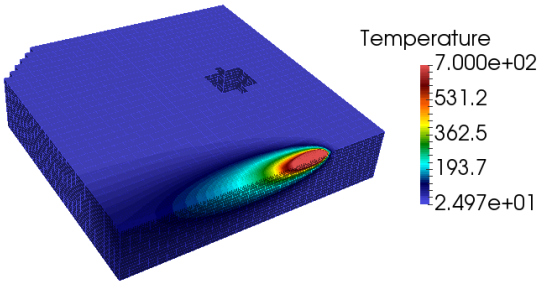
Results



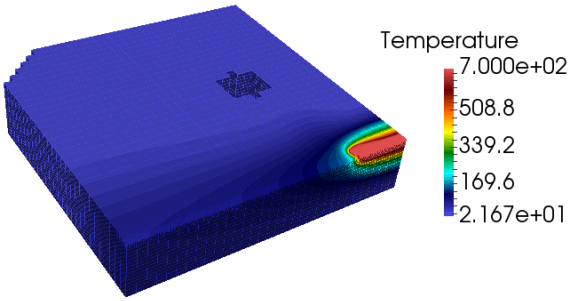
(a) Time step 20



(b) Time step 50



(c) Time step 100



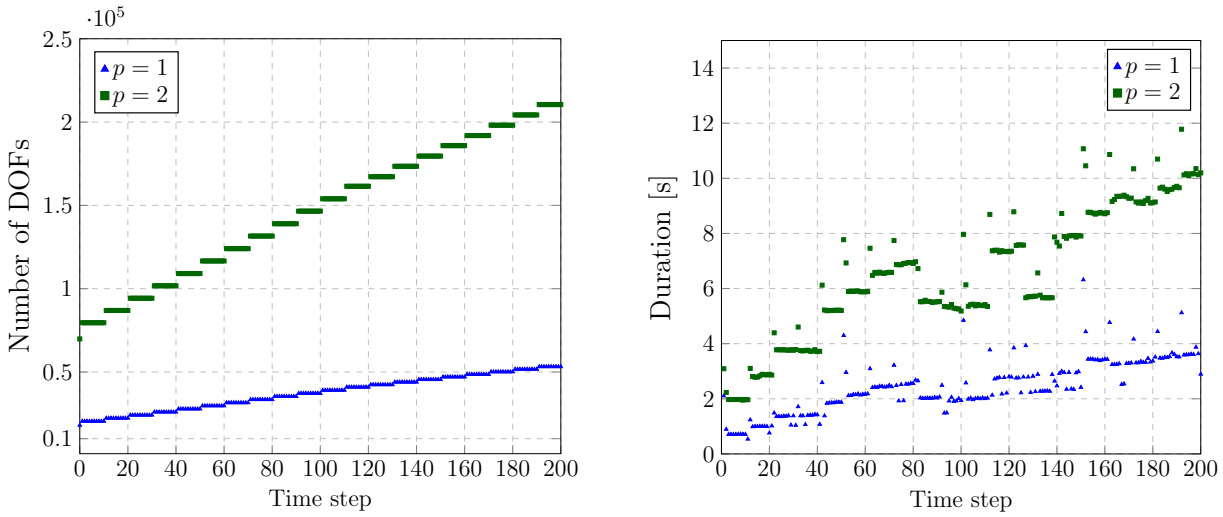
(d) Time step 200

Figure 6.6: Temperature distribution in the aluminum specimen at selected time steps.

The temperature distribution in the aluminum specimen for four different time steps is shown in Figure 6.6. A threshold is applied at the mean melting temperature of the aluminum alloy in order to visualize the geometry of the melt pool, which is represented by the bright red region in the figures. Figure 6.7 shows the computational cost of each time step for the two different discretizations used. For the sake of brevity, the number of processors used in the simulation is not varied. A total of 56 cores partitioned into 8 MPI tasks each using 4 OpenMP threads are used in both simulations. Figure 6.7a shows the number of DOFs per time step while the total duration of each time step is shown in Figure 6.7b. No mesh coarsening is applied in the example at hand, which explains the increase in the number of DOFs over time. This also leads to a gradual increase in the computational cost of the time steps as shown in Figure 6.7b. The duration of each time step is, however, still remarkably low considering the fact that an average of three Newton-Rhapson iterations are performed per time step. The total time needed to perform the 200 time steps on 56 cores is 10min for the $p = 1$ mesh and 23min 48s for the $p = 2$ discretization.

The current numerical example illustrates how efficient thermal SLM computations can be performed using the finite cell method and multi-level *hp*-refinement. Although the computation times shown are impressive, it would require an inordinate number of time steps (10 000 time steps per layer $\times$ 125 layer $\approx 1.25 \cdot 10^6$ time steps) to simulate the fabrication of the

entire aluminum cube that is only a mere 125 mm^3.



(a) Number of DOFs.

(b) Computational cost of the simulation time steps.

Figure 6.7: Number of DOFs and computational cost of the time steps in the high-fidelity thermal simulation on an aluminum specimen.

6.2.4 A layer-by-layer element activation approach

Simulating the fabrication of an entire artifact using high-fidelity SLM simulations is a tremendously challenging task due to the spatial and temporal scales that need to be resolved. Instead of explicitly modeling the position of the laser, it is possible to use simulation strategies that average the amount of energy input over a period of time. These approaches allow the computation of entire parts in a feasible amount of time as large time steps can be applied. Such strategies include a hatch-by-hatch approach or a layer-by-layer approach as presented in e.g. [Chiumenti et al., 2017]. In this thesis, a novel layer-by-layer approach based on the finite cell method is applied. Figure 4.11 illustrates how the proposed scheme can be used in a parallel setting. The following section briefly explains the central procedures in the suggested layer-by-layer approach. These include, how the averaged heat source Q is determined, the applications of the boundary conditions and manner in which time integration is performed. A detailed explanation on these subjects can be found in [Carraturo et al., 2020].

Heat Source Q

The heat source Q in layerwise SLM simulations is modeled as an equivalent heat source contribution applied as a volumetric load in the HAV (Heat-affected volume). It is directly proportional to the material's absorptivity η_t, the nominal power of the laser beam P and inversely proportional to the heat-affected volume and can be computed using the expression

$$Q = \frac{\eta_t P}{HAV}. \tag{6.13}$$

Boundary conditions

The Neumann boundary conditions applied in the layer-by-layer simulations correspond to those mentioned in Section 6.2.1 and include *i)* the conduction of heat between the lateral surfaces of the part and the surrounding powder, and *ii)* the combined heat loss by convection and radiation from the upper surface of the powder bed.

Time integration

As previously mentioned, time integration is performed using a backward Euler scheme. The size of a given time step Δt_h is not fixed but varies depending on whether a heating or cooling phase is been simulated. In the heating phase, the time step size is computed at runtime using the laser hatch distance h_d, laser velocity v and the surface activated within the time step HAS_i as

$$\Delta t_h = \sum_{i=1}^{n_{\text{layers}}} \frac{HAS_i}{v\, h_d}, \tag{6.14}$$

where HAS_i denotes the heat-affected surface of the i^{th} layer and n_{layers} the number of layers activated within the heating phase. This calculated time step corresponds to the amount of time it would take a laser to print the HAV under the assumption of a constant laser velocity during the printing process. The time step size for the cooling phase takes into account the time spent by the machine for recoating and time spent to print other components on the base plate. These values are user-defined and can be specified before the simulation and read in from an external input file.

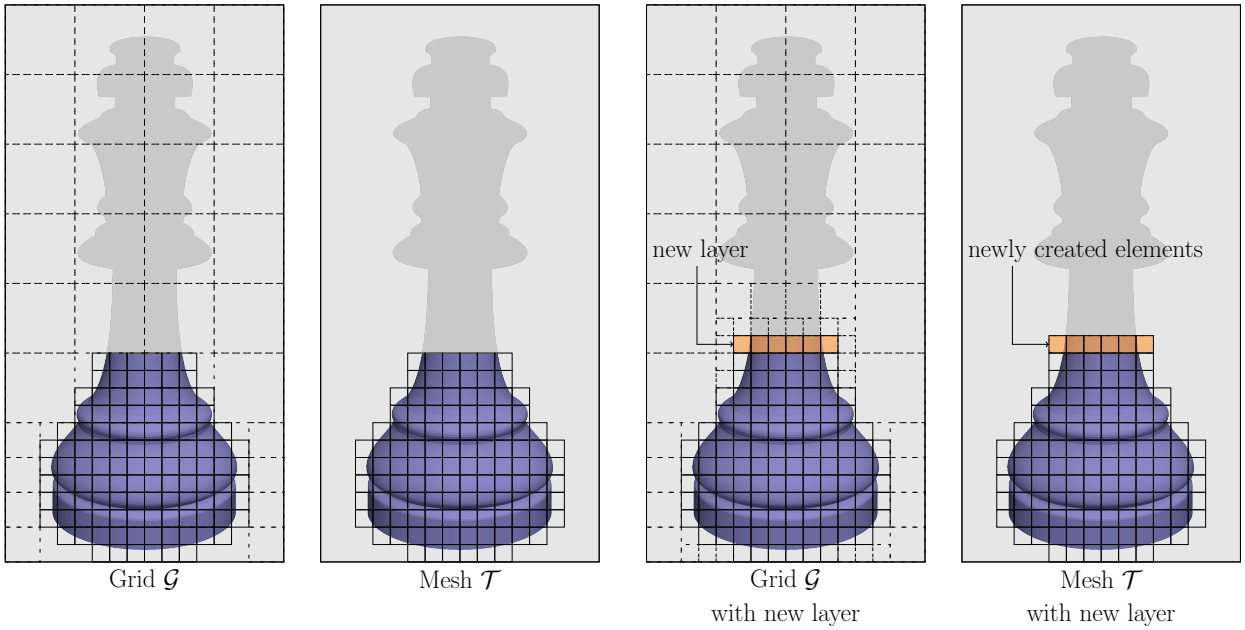
Layerwise activation scheme

An m-layers-by-m-layers activation scheme is applied in this work in which each finite cell contains m powder layers. The proposed scheme is an extension of the serial approach employed in [Carraturo, 2019; Carraturo et al., 2020] and can be used in a distributed-memory setting. It utilizes the finite cell method to model the growth of a component to be print through the evolution of the physical domain Ω_{phys}. The current physical domain $\Omega_{\text{phys}}(t)$ is obtained by intersecting the geometry of the complete artifact with a cuboid whose height is defined by the z-coordinate of the uppermost powder layer. This procedure is shown in Figure 6.8 for a chess piece geometry. The finite cell method decouples the geometric description from the discretization and enables the suggested layer-by-layer methodology to be used for the analysis of artifacts with complex geometries. Moreover, FCM allows the activation of multiple voxel layers within a single element. This is in contrast to standard m-layers-by-m-layers schemes that can only handle the activate of element layers.

Note that the proposed scheme uses a parallel adaptive Cartesian grid that is independent of the computational mesh to describe the geometry of the structure as elaborated in Section 4.4.1. This grid allows elements in a newly activated layer to be created "just-in-time" and no inactive elements need to be stored as in the quiet element method [Michaleris, 2014]. This "on-the-fly" activation strategy is chosen due to its suitability for parallel computations and its low memory requirements. During a heating time step the following steps are executed before solving the system of equations:

1. Update the physical domain $\Omega_{\text{phys}}(t)$ with respect to the current powder layer.
2. Create a new layer of elements if the $\Omega_{\text{phys}}(t)$ is not part of the computational mesh.
3. Perform a load balancing setup and transfer data between the MPI processes.
4. Create the external surfaces of the artifact that are needed for the application of boundary conditions using a marching cube algorithm.

A predefined number of cooling time steps is performed after every heating time step. It should be noted that no redistribution of the computational domain is required during the cooling phases.



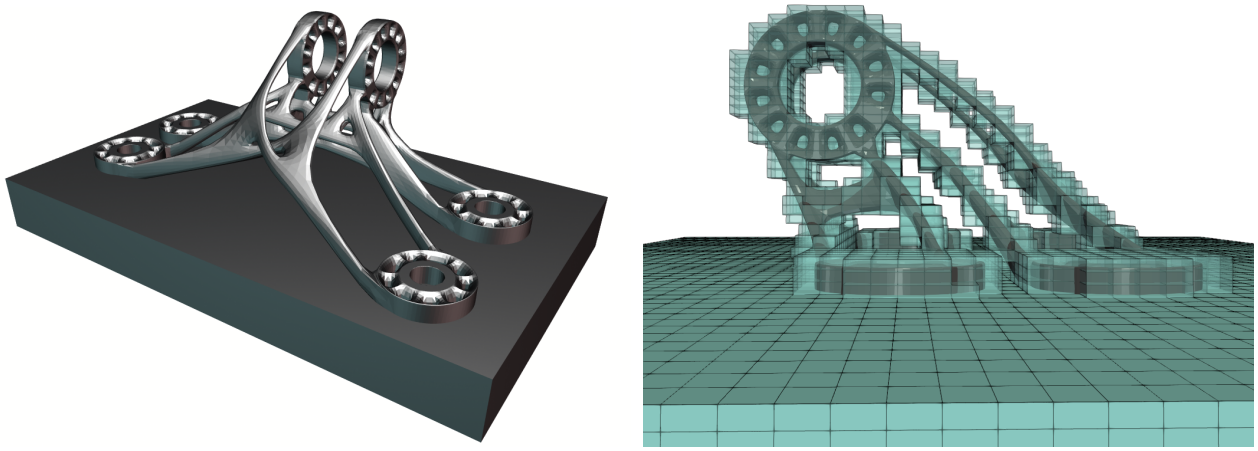
(a) Time step $t = 11$ (b) Time step $t = 12$

Figure 6.8: Illustration of the element creation procedure in m-layer-by-m-layer simulations. A new element layer is generated from an adaptive Cartesian grid at the beginning of every heating time step. This procedure circumvents the need of storing inactive elements.

6.2.4.1 Simulation of an optimized engine Bracket

The following section illustrates how the suggested layerwise methodology can be used to simulate the fabrication of an industrial component. The specimen considered is shown in Figure 6.9a and comprises of an optimized engine bracket, a commonly used model for AM applications.

A layer-by-layer thermal analysis of the fabrication process is performed using the shown geometry. The simulation consists of 60 stages, with each stage made up of a single heating step and cooling step, resulting in a total of 120 time steps. Each finite cell is divided into 5^3 voxels. $m = 5$ powder layers are activated within a single heating step. The mesh elements are generated in parallel using the approach presented in Section 4.4.1. A coarse initial grid $\mathcal{G}_{\text{init}}$ that spans the entire domain comprising $60 \times 100 \times 72$ grid cells is used for the element generation. Mesh elements belonging to a newly activated grid layer are created at the beginning of each heating time step following the approach illustrated in Figure 6.8. The number



(a) Geometry of an optimized engine bracket.

(b) Finite cell mesh.

Figure 6.9: Geometry and discretization of the engine bracket.

of DOFs for the two different discretizations ranges between 80 473 and 91 277 for the $p = 1$ mesh and between 313 087 and 337 529 for the $p = 2$ mesh.

In the example at hand, the performance of the hybrid parallel framework in transient nonlinear problems is assessed for two different polynomial orders, $p = \{1, 2\}$. A parallel Conjugate Gradient solver that utilizes the elementwise additive Schwarz preconditioning technique presented in Section 5.2 is used for the solution of the arising linear systems. In this study, the total simulation time for different hybrid computations is compared. The computations are run on the Linux cluster at the Leibniz Supercomputing Center, in Garching Germany, that is equipped with dual-socket Haswell nodes comprising 28 cores and 64GB of DDR4 memory per node. In each simulation, the 28 cores available on each node are partitioned into four MPI tasks and 7 OpenMP threads per task.

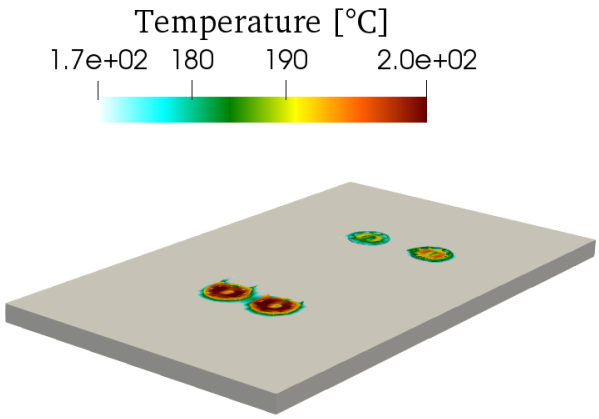
p	28 cores	56 nodes	112 cores	224 cores
1	23 min 01s	16 min 06s	12 min 06s	9 min 44s
2	40 min 08s	25 min 12s	20 min 57s	13 min 02s

Table 6.4: Comparison of the total runtime in the engine bracket example. The simulation consists of 120 time steps.

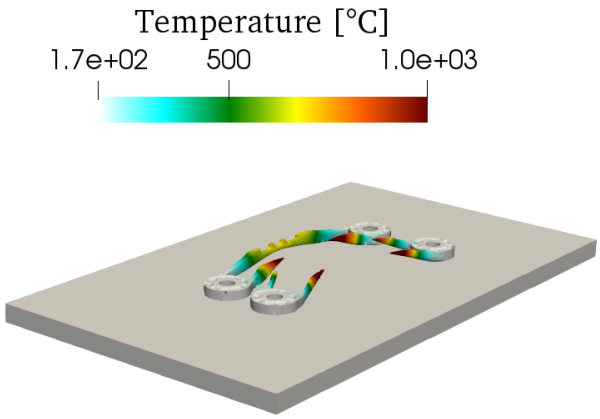
Table 6.4 shows a summary of the total execution time of simulations with different processor counts. One can see that increasing the number of processes leads to an improvement in the overall computation time. Figure 6.10 shows exemplary results of the temperature distribution in the bracket for selected time steps.

It should be noted that the proposed layer-by-layer approach cannot accurately capture the thermal history of points in the component due to the averaging performed. It can,

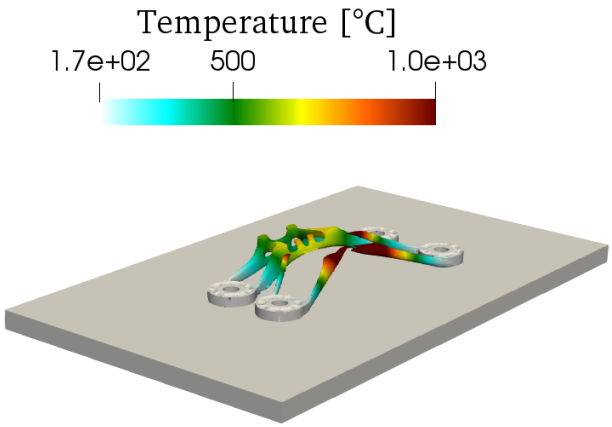
nevertheless, correctly predict regions in which overheating can occur, e.g. thin regions with downwards facing surfaces, see time steps 21 and 41 in Figure 6.10. This behavior occurs since no supporting structures are incorporated into the simulation. The current example is chosen to illustrate how the finite cell method can be used in part-scale simulations. The results shown are promising and can be extended by incorporating mesh refinement using the multi-level *hp*-method. Another interesting aspect that can be addressed in future work is the partial activation of elements in the build direction as proposed in [Carraturo et al., 2020].



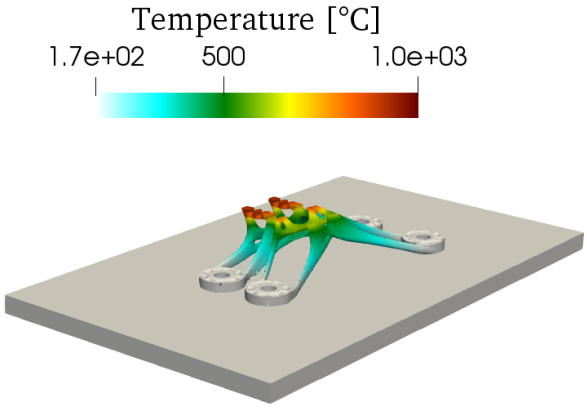
(a) Time step 1.



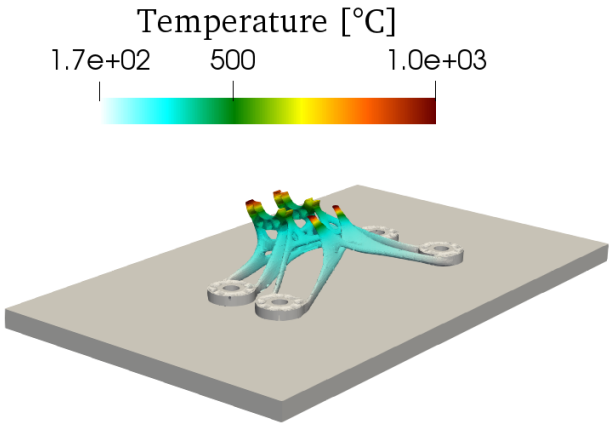
(b) Time step 21.



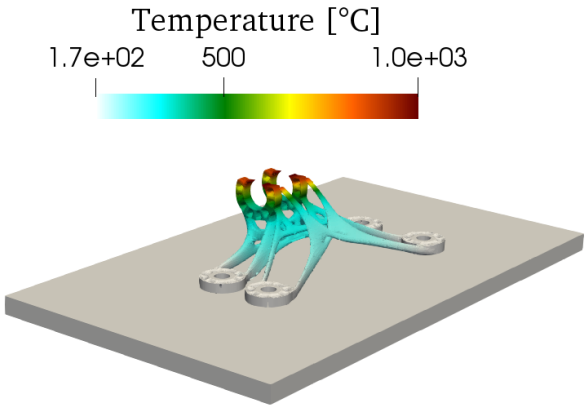
(c) Time step 41.



(d) Time step 61.



(e) Time step 81.



(f) Time step 101.

Figure 6.10: Distribution of the temperature in the GE-bracket for different heating time steps.

Chapter 7

Conclusion and outlook

The main focus of this thesis was the development of efficient algorithms and data structures for scalable finite cell analyses involving multi-level *hp*-refined grids on parallel computer systems. A simulation framework is presented that enables the use of FCM in different engineering applications of practical relevance. The developed methods can be readily applied in both linear and non-linear simulations and not only lead to faster execution times but also extend the boundary of computable FCM systems.

Summary

Two central aspects necessary for achieving satisfactory performance in distributed-memory finite cell analysis namely, the efficient handling of *hp*-grids and the development of robust and scalable iterative solvers for large finite cell systems, are addressed in this work. The following section provides a summary of these central topics.

Efficient mesh management

In this thesis, two strategies are presented for the management of parallel grid data structures. The first approach is based on a replication of the mesh on all MPI tasks. This strategy allows the mesh to be easily set up and significantly simplifies the management of degrees of freedom across processors. The method, however, has limited scalability as its high memory requirements render it applicable to small computing clusters with a large amount of memory per node. The second mesh management strategy was designed to remedy the shortcomings of the first approach and is based on a light-weight adaptive Cartesian grid that provides an abstraction of the computational mesh. Partitioning and load balancing operations are first performed on this data structure before being related to the mesh elements. The adaptive grid enables a distributed storage of mesh elements and has excellent parallel scalability as the amount of data stored by a single MPI task scales inversely with the number of MPI tasks. Since no single process knows the complete extent of the final computational mesh, special algorithms are needed to ensure parallel mesh compatibility, i.e. the DOF numbering across processor boundaries must be consistent. An algorithm that performs these operations in a fast manner for immersed multi-level *hp*-grids is presented in this work. This thesis portrays

how the suggested mesh management strategy allows finite cell computations with multiple millions and billions of DOFs on up to 32 000 cores.

Parallel mesh management in the context of dynamic meshes is also tackled in this thesis. A redistribution and rebalancing algorithm is proposed that maps data between MPI tasks when the computational load changes during a simulation. This algorithm is applied to evolving domains due to dynamic mesh refinement and growing domains. An application area that benefits from this approach is the field of additive manufacturing process simulation.

Robust iterative solvers for finite cell systems

One cornerstone of the present work is the development of dedicated preconditioners and robust solvers for finite cell systems. The pioneering work of de Prenter et. al on additive Schwarz preconditioner for immersed methods on uniform meshes is extended to FCM problems involving multi-level *hp*-refined grids. A key aspect of this preconditioning technique is the appropriate grouping of basis functions into additive Schwarz blocks. A pragmatic approach is adopted in this thesis that analyzes three different manners in which basis functions can be grouped. Basis functions are selected based on either leaf elements, base elements or element patches. Selecting additive Schwarz blocks based on leaf elements is the simplest approach and improves the conditioning of systems when truncation is applied. The preconditioner that is constructed in this way maintains the sparsity structure of the system matrix and is cheap to construct. It is shown, however, that scenarios can occur in which certain problematic modes are not detected by the preconditioner and "slip" through it. These modes lead to an increase in the number of iterations. Single-grid solvers, like the Conjugate Gradient method, are generally able to sufficiently deal with a few small modes that slip through the preconditioner as they only required a few additional iterations. Investing more effort in the construction of additive Schwarz preconditioner by utilizing blocks based on base elements and element patches can be applied to further improve the conditioning.

An *hp*-multigrid approach that leverages the hierarchical nature of the integrated Legendre polynomials and *h*-refinement scheme in the multi-level *hp*-method is presented in this thesis. The approach is combined with the finite cell method resulting in solvers with convergence rates that are independent of the cut configurations, mesh size and in some cases even the polynomial order and refinement level. The elementwise and patchwise Additive Schwarz techniques are applied as smoothers in the multigrid algorithm.

The efficiency of the developed solvers is showcased in a variety of benchmark problems and also shown in problems of practical industrial relevance. All solvers are implemented in an efficient numerical code and can be applied to systems with up to a few billion unknowns.

Additive manufacturing product and process simulation

The parallel mesh data structures developed in Chapter 4 and the preconditioning strategies for the finite cell method proposed in Chapter 5 are applied in the simulation of additive manufacturing product and process simulation. Using the parallel framework, it is possible to compute systems with multiple million and even billions of unknowns that can be used to analyze the material properties of additively-manufactured components. Two methods for performing thermal simulations of the selective laser melting process are presented in the final part of Chapter 6.

Outlook and future work

The methods presented in this thesis open the door for performing efficient computations of real-life engineering problems using the finite cell method, multi-level *hp*-refinement and parallel computing. Although a lot of time was invested in the design, implementation and testing of the code framework and developed algorithms, the presented work can be further improved and extended. Furthermore, several questions and undiscovered opportunities remain which can be addressed in future research projects.

The finite cell problems considered in this work embedded their physical domain in an axis-aligned Cartesian grid. The second mesh management strategy proposed in Chapter 4 is tailored to multi-level *hp*-analyses on such grids. This strategy can, however, be extended to arbitrarily-oriented hexahedral meshes and even unstructured grids. This step would allow the parallel framework to be used in an even broader setting.

Several aspects regarding the developed solvers and preconditioners can also be extended in future studies. A more elaborate mathematical investigation of the selection of additive Schwarz blocks for multi-level *hp*-refinement can be performed to find the best trade-off between the spectral properties of the preconditioned system and the computational cost of the preconditioner. Alternative preconditioning methods e.g. multiplicative Schwarz techniques or non-overlapping domain decomposition methods are other possible directions for future research. The *hp*-multigrid approach proposed in this thesis can be made more efficient by applying a geometric multigrid approach to accelerate the convergence of the $p = 1$ coarse system. This step will be beneficial in applications where the convergence of the coarse problem is slow e.g. in computations involving thin-walled structures undergoing bending.

Chapter 6 focused on the use of the finite cell method and multi-level refinement in the simulation of additive manufacturing products and processes. The presented results focused on showing the computational efficiency of the developed parallel framework. More effort should be directed towards extending the parallel framework to be able to perform parallel thermo-mechanical SLM simulations. An ongoing project in conjunction with the University of Pavia investigates the use of m-layer-by-m-layer approaches for the prediction of residual stresses and the deflection of the final part upon its removal from the base plate. A serial framework has been developed and validated against experimental measurements [Carraturo et al., 2020]. This framework can serve as a basis for future developments.

Improving the efficiency of `AdhoC++` is a perpetual task that cannot go without mentioning. A substantial amount of effort should be invested in improving the performance and scalability of the parallel framework developed in this thesis. General tasks include a thorough analysis of the node-level performance, the communication patterns and the identification and elimination of bottlenecks. Specific algorithms that need to be improved include the inversion of large additive Schwarz blocks for grids with several refinement levels and the load balancing strategy for grids with dynamic mesh refinements and different cut cell integration strategies. The optimization process must be guided by code optimization experts and involve the use of state-of-the-art performance profiling software. Code efficiency can be further improved by updating the third-party packages on which the parallel algorithms in `AdhoC++` are built upon. The results presented in this thesis were obtained using the following libraries, *i)* Trilinos Version 12.12, *ii)* IntelMPI compiler version 19.0, *iii)* GNU compiler 7.0, *iv)* ITK Version 5.12, *v)* BOOST Version 1.61. Updating this software to their most recent releases would able

users of `AdhoC++` to benefit from the improvements and new features in these libraries.

The methods developed in this thesis are applied to linear second-order elliptic problems and nonlinear simulations of metal additive manufacturing processes. Further application areas that can benefit for the proposed algorithms include the simulation of brittle fracture by means of a phase-field approach.

Bibliography

Ager, C., Schott, B., Winter, M., and Wall, W. (2019). A Nitsche-based cut finite element method for the coupling of incompressible fluid flow with poroelasticity. *Computer Methods in Applied Mechanics and Engineering*, 351:253–280.

Ahrens, J., Geveci, B., and Law, C. (2005). Paraview: An end-user tool for large data visualization. *Visualization Handbook*.

Amdahl, G. M. (1967). Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA. ACM.

Ammar, H., Samuel, A., and Samuel, F. (2008). Porosity and the fatigue behavior of hypoeutectic and hypereutectic aluminum-silicon casting alloys. *International Journal of Fatigue*, 30(6):1024–1035.

Anderson, R., Andrej, J., Barker, A., Bramwell, J., Camier, J.-S., Červený, J., Dobrev, V., Dudouit, Y., Fisher, A., Kolev, T., Pazner, W., Stowell, M., Tomov, V., Dahm, J., Medina, D., and Zampini, S. (2020). MFEM: A modular finite element methods library. *Computers & Mathematics with Applications*.

Annavarapu, C., Hautefeuille, M., and Dolbow, J. (2012). A robust Nitsche's formulation for interface problems. *Computer Methods in Applied Mechanics and Engineering*, 225–228:44–54.

Babuška, I. (1973). The Finite Element Method with Penalty. *Mathematics of Computation*, 27(122):221.

Babuška, I., Griebel, M., and Pitkäranta, J. (1989). The problem of selecting the shape functions for a p-type finite element. *International Journal for Numerical Methods in Engineering*, 28(8):1891–1908.

Babuška, I. and Suri, M. (1990). The p- and h-p versions of the finite element method, an overview. *Computer Methods in Applied Mechanics and Engineering*, 80(1):5–26.

Babuška, I., Szabo, B., and Katz, I. (1981). The p-Version of the Finite Element Method. *SIAM Journal on Numerical Analysis*, 18(3):515–545.

Badia, S., Martín, A. F., Neiva, E., and Verdugo, F. (2019). A generic finite element framework on parallel tree-based adaptive meshes. *preprint available on arXiv*.

Badia, S., Martín, A. F., Neiva, E., and Verdugo, F. (2020). The aggregated unfitted finite element method on parallel tree-based adaptive meshes. *preprint available on arXiv*.

Badia, S., Martín, A. F., and Principe, J. (2018a). FEMPAR: An Object-Oriented Parallel Finite Element Framework. In *Archives of computational methods in engineering*, volume 25, pages 195–271.

Badia, S. and Verdugo, F. (2017). Robust and scalable domain decomposition solvers for unfitted finite element methods. *Journal of Computational and Applied Mathematics*, 344:740–759.

Badia, S., Verdugo, F., and Martín, A. F. (2018b). The aggregated unfitted finite element method for elliptic problems. *Computer Methods in Applied Mechanics and Engineering*, 336:533–553.

Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F. (1997). Efficient management of parallelism in object oriented numerical software libraries. In Arge, E., Bruaset, A. M., and Langtangen, H. P., editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press.

Bangerth, W., Burstedde, C., Heister, T., and Kornbichler, M. (2011). Algorithms and Data Structures for Massively Parallel Generic Adaptive Finite Element Codes. *ACM Transactions on Mathematical Software*, 38(2):14:1–14:28.

Bangerth, W., Hartmann, R., and Kanschat, G. (2007). deal.II – a General Purpose Object Oriented Finite Element Library. *ACM Transactions on Mathematical Software*, 33(4):24/1–24/27.

Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and van der Vorst, H. (1994). *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM.

Bathe, K. J. (2007). *Finite Element Procedures*. Prentice Hall, New Jersey.

Béchet, E., Minnebo, H., Moës, N., and Burgardt, B. (2005). Improved implementation and robustness study of the X-FEM for stress analysis around cracks. *International Journal for Numerical Methods in Engineering*, 64(8):1033–1056.

Belytschko, T. and Black, T. (1999). Elastic crack growth in finite elements with minimal remeshing. *International Journal for Numerical Methods in Engineering*, 45(5):601–620.

Belytschko, T., Fish, J., and Bayliss, A. (1990). The spectral overlay on finite elements for problems with high gradients. *Computer Methods in Applied Mechanics and Engineering*, 81(1):71–89.

Berger-Vergiat, L., Waisman, H., Hiriyur, B., Tuminaro, R., and Keyes, D. (2012). Inexact Schwarz-algebraic multigrid preconditioners for crack problems modeled by extended finite element methods. *International Journal for Numerical Methods in Engineering*, 90(3):311–328.

Bog, T., Zander, N., Kollmannsberger, S., and Rank, E. (2017). Weak Imposition of Frictionless Contact Constraints on Automatically Recovered High-Order, Embedded Interfaces Using the Finite Cell Method. *Computational Mechanics*, 61(4):385–407.

Braess, D. (1986). On the combination of the multigrid method and conjugate gradients. In Hackbusch, W. and Trottenberg, U., editors, *Multigrid Methods II*, pages 52–64, Berlin, Heidelberg. Springer Berlin Heidelberg.

Braess, D. and Peisker, P. (1986). On the Numerical Solution of the Biharmonic Equation and the Role of Squaring Matrices for Preconditioning. *IMA Journal of Numerical Analysis*, 6(4):393–404.

Brenner, S. and Scott, L. (2008). *The Mathematical Theory of Finite Element Methods*. Springer.

Briggs, W., Henson, V., and McCormick, S. (2000). *A Multigrid Tutorial, 2nd Edition*. SIAM.

Burman, E. (2010). Ghost penalty. *Comptes Rendus Mathematique*, 348(21):1217–1220.

Burman, E. (2012). A Penalty-Free Nonsymmetric Nitsche-Type Method for the Weak Imposition of Boundary Conditions. *SIAM Journal on Numerical Analysis*, 50(4):1959–1981.

Burman, E., Claus, S., Hansbo, P., Larson, M. G., and Massing, A. (2014a). CutFEM: Discretizing geometry and partial differential equations. *International Journal for Numerical Methods in Engineering*, 104(7):472–501.

Burman, E., Claus, S., and Massing, A. (2014b). A Stabilized Cut Finite Element Method for the Three Field Stokes Problem. *SIAM Journal on Scientific Computing*, 37.

Burman, E., Elfverson, D., Hansbo, P., Larson, M. G., and Larsson, K. (2019). Cut topology optimization for linear elasticity with coupling to parametric nondesign domain regions. *Computer Methods in Applied Mechanics and Engineering*, 350:462–479.

Burman, E. and Hansbo, P. (2010). Fictitious domain finite element methods using cut elements: I. A stabilized Lagrange multiplier method. *Computer Methods in Applied Mechanics and Engineering*, 199(41–44):2680–2686.

Burman, E. and Hansbo, P. (2012). Fictitious domain finite element methods using cut elements: II. A stabilized Nitsche method. *Applied Numerical Mathematics*, 62(4):328–341.

Burstedde, C., Wilcox, L., and Ghattas, O. (2011). p4est : Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Scientific Computing*, 33:1103–1133.

Carey, G. F. and Barragy, E. (1989). Basis function selection and preconditioning high degree finite element and spectral methods. *BIT Numerical Mathematics*, 29:794–804.

Carraturo, M. (2019). *Modelling, Validation, and Design for Additive Manufacturing: Applications of numerical methods to 3D printing processes*. PhD Thesis, University of Pavia and the Technical University of Munich, Pavia and Munich.

Carraturo, M., Jomo, J., Kollmannsberger, S., Rank, E., Reali, A., and Auricchio, F. (2020). Modeling and experimental validation of an immersed thermo-mechanical part-scale analysis for laser powder bed fusion processes. *submitted to AM*.

Chandrasekhar, S. (1960). *Radiative Transfer*. Dover, New York.

Cheng, K. W. and Fries, T.-P. (2010). Higher-order XFEM for curved strong and weak discontinuities. *International Journal for Numerical Methods in Engineering*, 82(5):564–590.

Chern, I.-L. and Shu, Y.-C. (2007). A coupling interface method for elliptic interface problems. *Journal of Computational Physics*, 225(2):2138–2174.

Childs, H., Brugger, E., Whitlock, B., Meredith, J., Ahern, S., Pugmire, D., Biagas, K., Miller, M., Harrison, C., Weber, G. H., Krishnan, H., Fogal, T., Sanderson, A., Garth, C., Bethel, E. W., Camp, D., Rübel, O., Durant, M., Favre, J. M., and Navrátil, P. (2012). VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, pages 357–372.

Chiumenti, M., Neiva, E., Salsi, E., Cervera, M., Badia, S., Moya, J., Chen, Z., Lee, C., and Davies, C. (2017). Numerical modelling and experimental validation in Selective Laser Melting. *Additive Manufacturing*, 18:171–185.

Ciarlet, P. G. (2002). *The Finite Element Method for Elliptic Problems*. Number 40 in Classics in applied mathematics. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Clarke, L., Glendinning, I., and Hempel, R. (1994). The mpi message passing interface standard. In Decker, K. M. and Rehmann, R. M., editors, *Programming Environments for Massively Parallel Distributed Systems*, pages 213–218, Basel. Birkhäuser Basel.

Coll, A., Ribó, R., Pasenau, M., Escolano, E., Perez, J., Melendo, A., Monros, A., and Gárate, J. (2016). *GiD v.13 User Manual*.

Cottrell, J. A., Hughes, T. J. R., and Bazilevs, Y. (2009). *Isogeometric Analysis: Toward Integration of CAD and FEA*. John Wiley & Sons.

Craig, A. W. and Zienkiewicz, O. C. (1985). A multigrid algorithm using a hierarchical finite element basis. In Paddon, D. J. and Holstein, editors, *Multigrid Methods for Integral and Differential Equations*, pages 310–312. Clarendon Press, Oxford,.

Dagum, L. and Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55.

D'Angella, D., Zander, N., Kollmannsberger, S., Frischmann, F., Rank, E., Schröder, A., and Reali, A. (2016). Multi-level hp-adaptivity and explicit error estimation. *Advanced Modeling and Simulation in Engineering Sciences*, 3(1):33.

Dauge, M., Düster, A., and Rank, E. (2015). Theoretical and Numerical Investigation of the Finite Cell Method. *Journal of Scientific Computing*, 65(3):1039–1064.

de la Riva, A. P., Rodrigo, C., and Gaspar, F. J. (2019). A Robust Multigrid Solver for Isogeometric Analysis Based on Multiplicative Schwarz Smoothers. *SIAM J. Scientific Computing*, 41:321–345.

de Prenter, F., Verhoosel, C., and van Brummelen, E. (2019a). Preconditioning immersed isogeometric finite element methods with application to flow problems. *Computer Methods in Applied Mechanics and Engineering*, 348:604–631.

de Prenter, F., Verhoosel, C., Van Brummelen, H., Evans, J., Messe, C., Benzaken, J., and Maute, K. (2019b). Multigrid solvers for immersed finite element methods and immersed isogeometric analysis. *Computational Mechanics*, 65:807–838.

de Prenter, F., Verhoosel, C.V., van Zwieten, G.J., and van Brummelen, E.H. (2017). Condition number analysis and preconditioning of the finite cell method. *Computer Methods in Applied Mechanics and Engineering*, 316:297–327.

Dedner, A., Klöfkorn, R., Nolte, M., and Ohlberger, M. (2010). A generic interface for parallel and adaptive discretization schemes: abstraction principles and the dune-fem module. *Computing*, 90(3):165–196.

Demkowicz, L. (2007). *Computing with hp-Adaptive Finite Elements, Vol. 1: One and Two Dimensional Elliptic and Maxwell Problems*. Applied mathematics and nonlinear science series. Chapman & Hall/CRC, Boca Raton.

Devine, K., Boman, E., Heaphy, R., Hendrickson, B., and Vaughan, C. (2002). Zoltan Data Management Services for Parallel Dynamic Applications. *Computing in Science and Engineering*, 4(2):90–97.

Divi, S. C., Verhoosel, C. V., Auricchio, F., Reali, A., and van Brummelen, E. H. (2020). Error-estimate-based Adaptive Integration For Immersed Isogeometric Analysis. *Computers & Mathematics with Applications*.

Dongarra, J. J., Duff, L. S., Sorensen, D. C., and Vorst, H. A. V. (1998). *Numerical Linear Algebra for High Performance Computers*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Duczek, S., Berger, H., and Gabbert, U. (2015). The Finite Pore Method: a new approach to evaluate gas pores in cast parts by combining computed tomography and the finite cell method. *International Journal of Cast Metals Research*, 28(4):221–228.

Düster, A., Parvizian, J., Yang, Z., and Rank, E. (2008). The Finite Cell Method for Three-Dimensional Problems of Solid Mechanics. *Computer Methods in Applied Mechanics and Engineering*, 197(45–48):3768–3782.

Düster, A., Rank, E., and Szabó, B. A. (2017). The P-Version of the Finite Element Method and Finite Cell Methods. In *Encyclopedia of Computational Mechanics*, volume 2, pages 1–35. John Wiley & Sons, Chichester, West Sussex.

Eisenträger, S., Atroshchenko, E., and Makvandi, R. (2020). On the condition number of high order finite element methods: Influence of p-refinement and mesh distortion. *Computers & Mathematics with Applications*. Article in press, online version available.

Elfverson, D., Larson, M. G., and Larsson, K. (2018). CutIGA with basis function removal. *Advanced Modeling and Simulation in Engineering Sciences*, 5(1):6.

Elhaddad, M., Zander, N., Bog, T., Kudela, L., Kollmannsberger, S., Kirschke, J., Baum, T., Ruess, M., and Rank, E. (2018). Multi-level hp-finite cell method for embedded interface problems with application in biomechanics. *International Journal for Numerical Methods in Biomedical Engineering*, 34(4):e2951.

Embar, A., Dolbow, J., and Harari, I. (2010). Imposing Dirichlet boundary conditions with Nitsche's method and spline-based finite elements. *International Journal for Numerical Methods in Engineering*, 83(7):877–898.

Falgout, R. D., Jones, J. E., and Yang, U. M. (2006). The Design and Implementation of hypre, a Library of Parallel High Performance Preconditioners. In Bruaset, A. M. and Tveito, A., editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, pages 267–294, Berlin, Heidelberg. Springer Berlin Heidelberg.

Farhat, C. and Crivelli, L. (1989). A general approach to nonlinear FE computations on shared-memory multiprocessors. *Computer Methods in Applied Mechanics and Engineering*, 72(2):153–171.

Ferencz, R. and Hughes, T. (1998). *Solutions in Nonlinear Solid Mechanics*. Elsevier.

Fernández-Méndez, S. and Huerta, A. (2004). Imposing Essential Boundary Conditions in Mesh-Free Methods. *Computer Methods in Applied Mechanics and Engineering*, 193(12-14):1257–1275.

Ferronato, M. (2012). Preconditioning for Sparse Linear Systems at the Dawn of the 21st Century: History, Current Developments, and Future Perspectives. *International Scholarly Research Notices Applied Mathematics*, 2012:49.

Foresti, S., Brussino, G., Hassanzadeh, S., and Sonnad, V. (1989). Multilevel solution of the p-version of finite elements. *Computer Physics Communications*, 53:349–355.

Foster, I. (1995). *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Gahalaut, K., Kraus, J., and Tomar, S. (2013). Multigrid methods for isogeometric discretization. *Computer Methods in Applied Mechanics and Engineering*, 253:413–425.

Gerstenberger, A. and Wall, W. A. (2010). An embedded Dirichlet formulation for 3D continua. *International Journal for Numerical Methods in Engineering*, 82(5):537–563.

Giovannelli, L., Ródenas, J. J., Navarro-Jimenez, J. M., and Tur, M. (2014). Element stiffness matrix integration in image-based cartesian grid finite element method. In Zhang, Y. J. and Tavares, J. M. R. S., editors, *Computational Modeling of Objects Presented in Images.*

Fundamentals, Methods, and Applications, pages 304–315, Cham. Springer International Publishing.

Glowinski, R. and Kuznetsov, Y. (2007). Distributed lagrange multipliers based on fictitious domain method for second order elliptic problems. *Computer Methods in Applied Mechanics and Engineering*, 196(8):1498–1506.

Goldak, J., Chakravarti, A., and Bibby, M. (1984). A new finite element model for welding heat sources. *Metallurgical Transactions B*, 15(2):299–305.

Griebel, M. and Schweitzer, M. A. (2003). A Particle-Partition of Unity Method Part V: Boundary Conditions. In Hildebrandt, S. and Karcher, H., editors, *Geometric Analysis and Nonlinear Partial Differential Equations*, pages 519–542. Springer, Berlin, Heidelberg.

Gustafson, J. L. (1988). Reevaluating Amdahl's Law. *Commun. ACM*, 31(5):532–533.

Hackbusch, W. (1994). *Iterative solution of large sparse systems of equations (1st ed.)*, volume 95. Springer Switzerland.

Hackbusch, W. (2013). *Multi-Grid Methods and Applications*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg.

Hackbusch, W. and Trottenberg, U. (1982). *Multigrid Methods. Proceedings of the Conference Held at Köln-Porz, November 23-27, 1981*.

Hager, G. and Wellein, G. (2010). *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., Lehoucq, R. B., Long, K. R., Pawlowski, R. P., Phipps, E. T., Salinger, A. G., Thornquist, H. K., Tuminaro, R. S., Willenbring, J. M., Williams, A., and Stanley, K. S. (2005). An overview of the Trilinos project. *ACM Transactions on Mathematical Software*, 31(3):397–423.

Hiriyur, B., Tuminaro, R., Waisman, H., Boman, E., and Keyes, D. (2012). A Quasi-algebraic Multigrid Approach to Fracture Problems Based on Extended Finite Elements. *Siam Journal on Scientific Computing*, 34(2):603–626.

Hofreither, C., Jüttler, B., Kiss, G., and Zulehner, W. (2016). Multigrid methods for isogeometric analysis with thb-splines. *Computer Methods in Applied Mechanics and Engineering*, 308:96–112.

Höllig, K., Apprich, C., and Streit, A. (2005). Introduction to the Web-method and its applications. *Advances in Computational Mathematics*, 23(1):215–237.

Höllig, K., Reif, U., and Wipper, J. (2001). Weighted Extended B-Spline Approximation of Dirichlet Problems. *SIAM Journal on Numerical Analysis*, 39(2):442–462.

Hu, N., Guo, X., and Katz, I. (1998). Bounds for eigenvalues and condition numbers in the p-version of the finite element method. *Math. Comput.*, 67:1423–1450.

Hubrich, S., Di Stolfo, P., Kudela, L., Kollmannsberger, S., Rank, E., Schröder, A., and Düster, A. (2017). Numerical integration of discontinuous functions: moment fitting and smart octree. *Computational Mechanics*, 60:863–881.

Hug, L., Kollmannsberger, S., Yosibash, Z., and Rank, E. (2020). A 3D benchmark problem for crack propagation in brittle fracture. *Computer Methods in Applied Mechanics and Engineering*, 364:112905.

Hughes, T. (2000). *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Civil and Mechanical Engineering. Dover Publications.

Ibanez, L., Schroeder, W., Ng, L., and Cates, J. (2003). *The ITK Software Guide*. Kitware, Inc., first edition. ISBN 1-930934-10-6.

Intel (2009). *Intel Math Kernel Library. Reference Manual*. Intel Corporation.

Jomo, J., de Prenter, F., Elhaddad, M., D'Angella, D., Verhoosel, C., Kollmannsberger, S., Kirschke, J., Nübel, V., van Brummelen, E., and Rank, E. (2019). Robust and parallel scalable iterative solutions for large-scale finite cell analyses. *Finite Elements in Analysis and Design*, 163:14–30.

Jomo, J. N., Zander, N., Elhaddad, M., Özcan, A., Kollmannsberger, S., Mundani, R.-P., and Rank, E. (2017). Parallelization of the multi-level hp-adaptive finite cell method. *Computers & Mathematics with Applications*, 74(1):126–142.

Joulaian, M., Duczek, S., Gabbert, U., and Düster, A. (2014). Finite and Spectral Cell Method for Wave Propagation in Heterogeneous Materials. *Computational Mechanics*, 54(3):661–675.

Joulaian, M., Hubrich, S., and Düster, A. (2016). Numerical Integration of Discontinuities on Arbitrary Domains Based on Moment Fitting. *Computational Mechanics*, 57:979–999.

JP1 (2016). Chess Display Pieces - King & Queen . `http://www.thingiverse.com/thing:1661934`.

Kaasschieter, E. F. (1988). Preconditioned Conjugate Gradients for Solving Singular Systems. *J. Comput. Appl. Math.*, 24(1-2):265–275.

Kamensky, D., Hsu, M.-C., Schillinger, D., Evans, J., Aggarwal, A., Bazilevs, Y., Sacks, M., and Hughes, T. (2015). An immersogeometric variational framework for fluid-structure interaction: Application to bioprosthetic heart valves. *Computer Methods in Applied Mechanics and Engineering*, 284:1005–1053.

Keaveny, T. M., Morgan, E. F., Niebur, G. L., and Yeh, O. C. (2001). Biomechanics of Trabecular Bone. *Annual Review of Biomedical Engineering*, 3(1):307–333.

Kettler, R. (1982). Analysis and comparison of relaxation schemes in robust multigrid and preconditioned conjugate gradient methods. In Hackbusch, W. and Trottenberg, U., editors, *Multigrid Methods*, pages 502–534, Berlin, Heidelberg. Springer.

King, W., Anderson, A., Ferencz, R., Hodge, N., Kamath, C., and Khairallah, S. (2015). Overview of modelling and simulation of metal powder bed fusion process at Lawrence Livermore National Laboratory. *Materials Science and Technology*, 31:957–968.

Kirk, B. S., Peterson, J. W., Stogner, R. H., and Carey, G. F. (2006). `libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers*, 22(3–4):237–254.

Kollmannsberger, S., Carraturo, M., Reali, A., and Auricchio, F. (2019). Accurate prediction of melt pool shapes in laser powder bed fusion by the non-linear temperature equation including phase changes. *Integrating Materials and Manufacturing Innovation*, 8:167–177.

Kollmannsberger, S., Özcan, A., Carraturo, M., Zander, N., and Rank, E. (2018). A hierarchical computational model for moving thermal loads and phase changes with applications to selective laser melting. *Computers & Mathematics with Applications*, 75(5):1483–1497.

Korshunova, N., Jomo, J., Lékó, G., Reznik, D., Kollmannsberger, S., Balázs, P., and Rank, E. (2020). From CT-scans to material characterization:linear elastic mechanical behavior. *accepted in Computers & Mathematics with Applications*.

Kudela, L., Zander, N., Bog, T., Kollmannsberger, S., and Rank, E. (2015). Efficient and accurate numerical quadrature for immersed boundary methods. *Advanced Modeling and Simulation in Engineering Sciences*, 2(1):1–22.

Kudela, L., Zander, N., Kollmannsberger, S., and Rank, E. (2016). Smart Octrees: Accurately Integrating Discontinuous Functions in 3D. *Computer Methods in Applied Mechanics and Engineering*, 306:406–426.

Larson, M. and Bengzon, F. (2013). *The Finite Element Method: Theory, Implementation, and Applications*. Springer.

Lehrenfeld, C. and Reusken, A. (2017). Optimal preconditioners for Nitsche-XFEM discretizations of interface problems. *Numerische Mathematik*, 135(2):313–332.

Li, J., Liao, W.-k., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., and Zingale, M. (2003). Parallel netCDF: A High-Performance Scientific I/O Interface. pages 39–39.

Lions, P. (1988). On the Schwarz alternating method. I. In *First international symposium on domain decomposition methods for partial differential equations*, pages 1–42.

Liu, N., Fu, J., Carothers, C., Sahni, O., Jansen, K., and Shephard, M. (2010). Massively Parallel I/O for Partitioned Solver Systems. *Parallel Processing Letters*, 20:377–395.

Logg, A. (2009). Efficient representation of computational meshes. *International Journal of Computational Science and Engineering*, 4(4):283–295.

Lowan, A., Davids, N., Levenson, A., and (U.S.), M. T. P. (1942). *Table of the Zeros of the Legendre Polynomials of Order 1-16 and the Weight Coefficients for Gauss' Mechanical Quadrature Formula*. Maestri e testimoni.

Main, A. and Scovazzi, G. (2018a). The shifted boundary method for embedded domain computations. Part I: Poisson and Stokes problems. *Journal of Computational Physics*, 372:972–995.

Main, A. and Scovazzi, G. (2018b). The shifted boundary method for embedded domain computations. Part II: Linear advection-diffusion and incompressible Navier-Stokes equations. *Journal of Computational Physics*, 372:996–1026.

Maitre, J. and Pourquier, O. (1996). Condition number and diagonal preconditioning: comparison of the p-version and the spectral element methods. *Numerische Mathematik*, 74:69–84.

Maple, C. (2003). Geometric design and space planning using the marching squares and marching cube algorithms. In *2003 International Conference on Geometric Modeling and Graphics, 2003. Proceedings*, pages 90–95.

Marussig, B., Hiemstra, R., and Hughes, T. J. (2018). Improved conditioning of isogeometric analysis matrices for trimmed geometries. *Computer Methods in Applied Mechanics and Engineering*, 334:79–110.

Massing, A., Schott, B., and Wall, W. (2018). A stabilized Nitsche cut finite element method for the Oseen problem. *Computer Methods in Applied Mechanics and Engineering*, 328:262–300.

Matsokin, A. and Nepomnyaschikh, S. (1985). The Schwarz alternation method in a subspace. *Soviet Mathematics (Izv. Vyssh. Uchebn. Zaved. Mat.)*.

Mayer, H., Papakyriacou, M., Zettl, B., and Stanzl-Tschegg, S. (2003). Influence of porosity on the fatigue limit of die cast magnesium and aluminium alloys. *International Journal of Fatigue*, 25(3):245–256.

Menk, A. and Bordas, S. (2011). A robust preconditioning technique for the extended finite element method. *International Journal for Numerical Methods in Engineering*, 85(13):1609–1632.

Michaleris, P. (2014). Modeling metal deposition in heat transfer analyses of additive manufacturing processes. *Finite Elements in Analysis and Design*, 86:51–60.

Mitchell, W. F. (2010). The hp-multigrid method applied to hp-adaptive refinement of triangular grids. *Numerical Linear Algebra with Applications*, 17(2-3):211–228.

Moës, N., Dolbow, J., and Belytschko, T. (1999). A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46(1):131–150.

Monavari, M. (2011). *The Finite Cell Method for the analysis of die casts based on X-ray computed tomography*. Master's thesis, Technische Universität München, Munich.

Moore, P. K. and Flaherty, J. E. (1992). Adaptive local overlapping grid methods for parabolic systems in two space dimensions. *Journal of Computational Physics*, 98(1):54–63.

Mote, C. D. (1971). Global-local finite element. *International Journal for Numerical Methods in Engineering*, 3(4):565–574.

Nadal Soriano, E., Ródenas, J., Albelda, J., Tur, M., Tarancón, J., and Fuenmayor, F. (2013). Efficient Finite Element Methodology Based on Cartesian Grids: Application to Structural Shape Optimization. *Abstract and Applied Analysis*, 2013.

Nagaraja, S., Elhaddad, M., Ambati, M., Kollmannsberger, S., De Lorenzis, L., and Rank, E. (2019). Phase-field modeling of brittle fracture with multi-level *hp*-fem and the finite cell method. *Computational Mechanics*, 63:1283–1300.

Navarro-Jiménez, J. M., Tur, M., Fuenmayor, F. J., and Ródenas, J. J. (2018). On the effect of the contact surface definition in the Cartesian grid finite element method. *Advanced Modeling and Simulation in Engineering Sciences*, 5(1):12.

Neiva, E. and Badia, S. (2020). Robust and scalable h-adaptive aggregated unfitted finite elements for interface elliptic problems. *preprint available on arXiv*.

Nicoletto, G., Anzelotti, G., and Konečná, R. (2010). X-ray computed tomography vs. metallography for pore sizing and fatigue of cast Al-alloys. *Procedia Engineering*, 2(1):547–554. Fatigue 2010.

Nitsche, J. (1971). Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 36(1):9–15.

Ollivier-Gooch, C., Diachin, L. A., Shephard, M. S., Tautges, T. J., Kraftcheck, J., Leung, V. J., Luo, X., and Miller, M. (2010). An Interoperable, Data-Structure-Neutral Component for Mesh Query and Manipulation. *ACM Transactions on Mathematical Software*, 37:29:1–29:28.

OpenMP Architecture Review Board (2008). OpenMP Application Program Interface Version 3.0.

Özcan, A., Kollmannsberger, S., Jomo, J., and Rank, E. (2019). Residual stresses in metal deposition modeling: Discretizations of higher order. *Computers & Mathematics with Applications*, 78(7):2247–2266.

Parvizian, J., Düster, A., and Rank, E. (2007). Finite Cell Method. *Computational Mechanics*, 41(1):121–133.

Paszyński, M. and Demkowicz, L. (2006). Parallel, fully automatic hp-adaptive 3D finite element package. *Engineering with Computers*, 22(3-4):255–276.

Paszyński, M. and Pardo, D. (2011). Parallel self-adaptive *hp* finite emelent method with shared data structure. *Computer Methods in Material Science*, 11(2):399–405.

Patra, A., Laszloffy, A., and Long, J. (2003). Data structures and load balancing for parallel adaptive hp finite-element methods. *Computers & Mathematics with Applications*, 46:105–123.

Rachowicz, W., Pardo, D., and Demkowicz, L. (2006). Fully automatic hp-adaptivity in three dimensions. *Computer Methods in Applied Mechanics and Engineering*, 195(37–40):4816–4842.

Rank, E. (1992). Adaptive remeshing and h-p domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 101(1–3):299–313.

Rank, E., Ruess, M., Kollmannsberger, S., Schillinger, D., and Düster, A. (2012). Geometric Modeling, Isogeometric Analysis and the Finite Cell Method. *Computer Methods in Applied Mechanics and Engineering*, 249-252:104–115.

Reinders, J. (2007). *Intel Threading Building Blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition.

Ruess, M., Schillinger, D., Bazilevs, Y., Varduhn, V., and Rank, E. (2013). Weakly enforced essential boundary conditions for NURBS-embedded and trimmed NURBS geometries on the basis of the finite cell method. *International Journal for Numerical Methods in Engineering*, 95(10):811–846.

Ruess, M., Schillinger, D., Özcan, A. I., and Rank, E. (2014). Weak Coupling for Isogeometric Analysis of Non-Matching and Trimmed Multi-Patch Geometries. *Computer Methods in Applied Mechanics and Engineering*, 269:46–71.

Ruess, M., Tal, D., Trabelsi, N., Yosibash, Z., and Rank, E. (2012). The finite cell method for bone simulations: verification and validation. *Biomechanics and modeling in mechanobiology*, 11(3-4):425–37.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition.

Schenk, O. and Gärtner, K. (2011). *PARDISO*, pages 1458–1464. Springer US, Boston, MA.

Schillinger, D., Dedè, L., Scott, M. A., Evans, J. A., Borden, M. J., Rank, E., and Hughes, T. J. (2012). An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of NURBS, immersed boundary methods, and T-spline CAD surfaces. *Computer Methods in Applied Mechanics and Engineering*, 249-252:116–150.

Schillinger, D. and Rank, E. (2011a). An Unfitted Hp-Adaptive Finite Element Method Based on Hierarchical B-Splines for Interface Problems of Complex Geometry. *Computer Methods in Applied Mechanics and Engineering*, 200(47-48):3358–3380.

Schillinger, D. and Rank, E. (2011b). Numerical Analysis of Lamb Waves Using the Finite and Spectral Cell Methods. *Computer Methods in Applied Mechanics and Engineering*, 99(47-48):3358–3380.

Schillinger, D. and Ruess, M. (2014). The Finite Cell Method: A Review in the Context of Higher-Order Structural Analysis of CAD and Image-Based Geometric Models. *Archives of Computational Methods in Engineering*, 22(3):391–455.

Schling, B. (2011). *The Boost C++ Libraries*. XML Press.

Schott, B. (2017). *Stabilized Cut Finite Element Methods for Complex Interface Coupled Flow Problems*. Dissertation, Technische Universität München, Munich.

Schott, B., Ager, C., and Wall, W. (2019). A monolithic approach to fluid-structure interaction based on a hybrid Eulerian-ALE fluid domain decomposition involving cut elements. *International Journal for Numerical Methods in Engineering*, 119(3):208–237.

Shapira, Y. (2003). *Matrix-Based Multigrid: Theory and Applications*. Numerical methods and algorithms. Kluwer Academic Publishers.

Shewchuk, J. R. (1994). An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, USA.

Smith, B., Bjørstad, P., and Gropp, W. (1996). *Domain Decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge university press.

Šolín, P. (2004). *Higher-Order Finite Element Methods*. Studies in advanced mathematics. Chapman & Hall/CRC, Boca Raton.

Šolín, P., Červený, J., and Doležel, I. (2008). Arbitrary-level hanging nodes and automatic adaptivity in the hp-FEM. *Mathematics and Computers in Simulation*, 77(1):117–132.

Šolín, P., Dubcova, L., and Doležel, I. (2010). Adaptive hp-FEM with arbitrary-level hanging nodes for Maxwell's equations. *Advances in Applied Mathematics and Mechanics*, 2(4):518–532.

Stenberg, R. (1995). On some techniques for approximating boundary conditions in the finite element method. *Journal of Computational and Applied Mathematics*, 63(1):139 – 148. Proceedings of the International Symposium on Mathematical Modelling and Computational Methods Modelling 94.

Sukumar, N., Moës, N., Moran, B., and Belytschko, T. (2000). Extended finite element method for three-dimensional crack modelling. *International Journal for Numerical Methods in Engineering*, 48(11):1549–1570.

Szabó, B. A. and Babuška, I. (1991). *Finite Element Analysis*. John Wiley & Sons, New York.

The HDF Group (1997). Hierarchical Data Format, version 5. `http://www.hdfgroup.org/HDF5/`.

Tielen, R., Möller, M., Göddeke, D., and Vuik, C. (2020). p-multigrid methods and their comparison to h-multigrid methods within Isogeometric Analysis. *Computer Methods in Applied Mechanics and Engineering*, 372:113347.

Toselli, A. and Widlund, O. (2005). *Domain Decomposition Methods: Algorithms and Theory*. Springer.

Trottenberg, U., Ulrich Trottenberg, C., Oosterlee, C., Schuller, A., Brandt, A., Oswald, P., and Stüben, K. (2001). *Multigrid*. Elsevier Science.

Tu, T., O'Hallaron, D. R., and Ghattas, O. (2005). Scalable parallel octree meshing for terascale applications. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, pages 4–, Washington, DC, USA. IEEE Computer Society.

Verdugo, F., Martín, A. F., and Badia, S. (2019). Distributed-memory parallelization of the aggregated unfitted finite element method. *Computer Methods in Applied Mechanics and Engineering*, 357:112583.

Verhoosel, C.V., van Zwieten, G.J., van Rietbergen, B., and de Borst, R. (2015). Image-Based Goal-Oriented Adaptive Isogeometric Analysis with Application to the Micro-Mechanical Modeling of Trabecular Bone. *Computer Methods in Applied Mechanics and Engineering*, 284:138–164.

Vinarcik, E. (2002). *High Integrity Die Casting Processes*. Wiley.

Waisman, H. and Berger-Vergiat, L. (2013). An adaptive domain decomposition preconditioner for crack propagation problems modeled by XFEM. *International Journal for Multiscale Computational Engineering*, 11(6):633–654.

Wesseling, P. (2004). *An Introduction to Multigrid Methods*. An Introduction to Multigrid Methods. R.T. Edwards.

Würkner, M., Duczek, S., Berger, H., Köppe, H., and Gabbert, U. (2018). A Software Platform for the Analysis of Porous Die-Cast Parts Using the Finite Cell Method. In Altenbach, H., Carrera, E., and Kulikov, G., editors, *Analysis and Modelling of Advanced Structures and Smart Systems*, volume 81 of *Advanced Structured Materials*, chapter 14, pages 327–341. Springer, Singapore, 1 edition.

Yang, Z., Ruess, M., Kollmannsberger, S., Düster, A., and Rank, E. (2012). An Efficient Integration Technique for the Voxel-Based Finite Cell Method. *International Journal for Numerical Methods in Engineering*, 91(5):457–471.

Yi, J. Z., Gao, Y. X., Lee, P. D., Flower, H. M., and Lindley, T. C. (2003). Scatter in fatigue life due to effects of porosity in cast A356-T6 aluminum-silicon alloys. *Metallurgical and Materials Transactions A*, 34(9):1879.

Yosibash, Z. (2011). *Singularities in Elliptic Boundary Value Problems and Elasticity and Their Connection with Failure Initiation*. Interdisciplinary Applied Mathematics. Springer New York.

Yserentant, H. (1985). Hierarchical bases of finite-element spaces in the discretization of nonsymmetric elliptic boundary value problems. *Computing*, 35:39–49.

Yserentant, H. (1986). Hierarchical bases give conjugate gradient type methods a multigrid speed of convergence. *Applied Mathematics and Computation*, 19(1):347–358.

Yushkevich, P. A., Piven, J., Cody Hazlett, H., Gimpel Smith, R., Ho, S., Gee, J. C., and Gerig, G. (2006). User-Guided 3D Active Contour Segmentation of Anatomical Structures: Significantly Improved Efficiency and Reliability. *Neuroimage*, 31(3):1116–1128.

Zander, N. (2017). *Multi-Level Hp-FEM: Dynamically Changing High-Order Mesh Refinement with Arbitrary Hanging Nodes*. PhD Thesis, Technische Universität München, Munich.

Zander, N., Bog, T., Elhaddad, M., Frischmann, F., Kollmannsberger, S., and Rank, E. (2016a). The Multi-Level hp-Method for Three-Dimensional Problems: Dynamically Changing High-Order Mesh Refinement with Arbitrary Hanging Nodes. *Computer Methods in Applied Mechanics and Engineering*, 310:252–277.

Zander, N., Bog, T., Kollmannsberger, S., Schillinger, D., and Rank, E. (2015). Multi-Level hp-Adaptivity: High-Order Mesh Adaptivity without the Difficulties of Constraining Hanging Nodes. *Computational Mechanics*, 55(3):499–517.

Zander, N., Ruess, M., Bog, T., Kollmannsberger, S., and Rank, E. (2016b). Multi-Level hp-Adaptivity for Cohesive Fracture Modeling. *International Journal for Numerical Methods in Engineering*, 109(13):1723–1755.