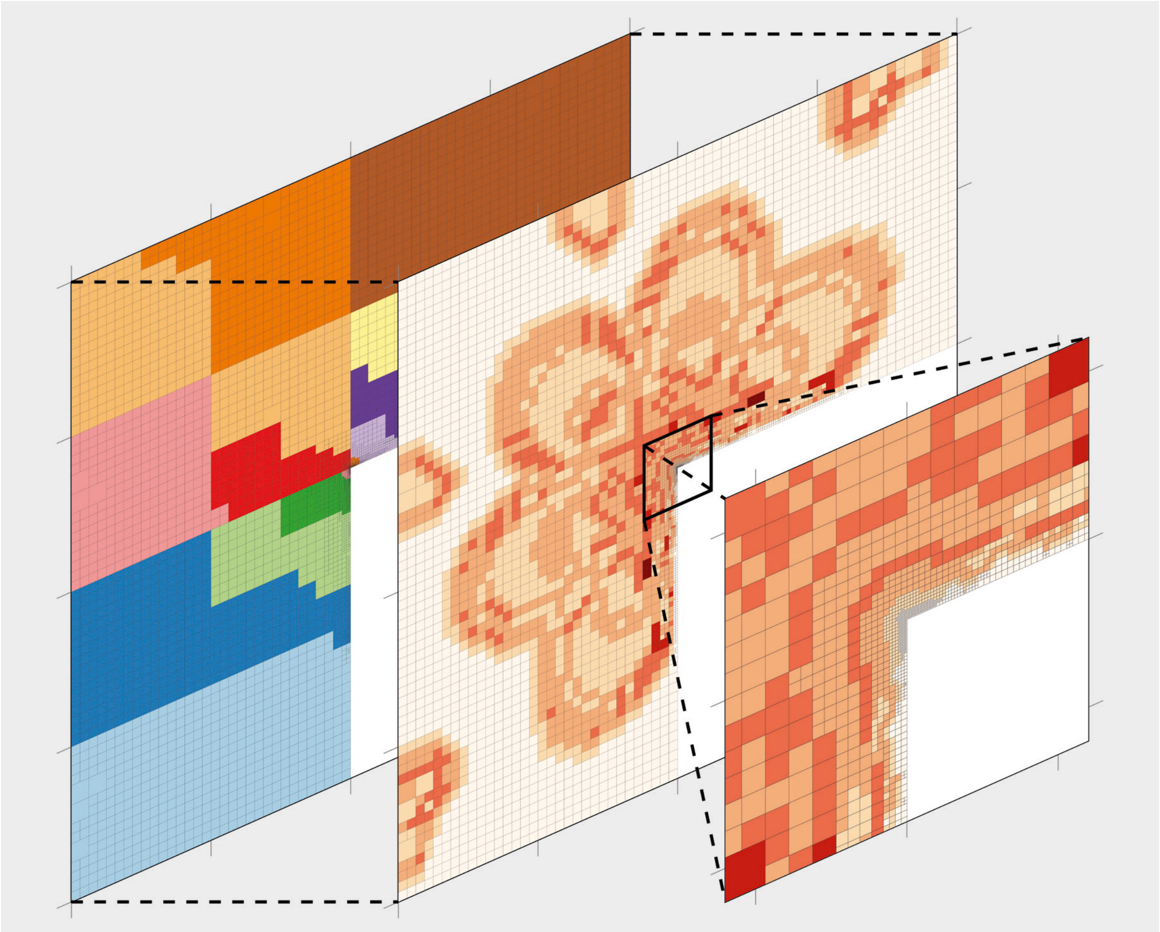


Algorithms for massively parallel generic *hp*-adaptive finite element methods

Marc Fehling

IAS Series
Band / Volume 43
ISBN 978-3-95806-486-7

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum

Schriften des Forschungszentrums Jülich
IAS Series Band / Volume 43

Forschungszentrum Jülich GmbH
Institute for Advanced Simulation (IAS)
Civil Safety Research (IAS-7)

Algorithms for massively parallel generic *hp*-adaptive finite element methods

Marc Fehling

Schriften des Forschungszentrums Jülich
IAS Series Band / Volume 43

ISSN 1868-8489 ISBN 978-3-95806-486-7

Bibliografische Information der Deutschen Nationalbibliothek.
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte Bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Herausgeber und Vertrieb:	Forschungszentrum Jülich GmbH Zentralbibliothek, Verlag 52425 Jülich Tel.: +49 2461 61-5368 Fax: +49 2461 61-6103 zb-publikation@fz-juelich.de www.fz-juelich.de/zb
Umschlaggestaltung:	Grafische Medien, Forschungszentrum Jülich GmbH
Druck:	Grafische Medien, Forschungszentrum Jülich GmbH
Copyright:	Forschungszentrum Jülich 2020

Schriften des Forschungszentrums Jülich
IAS Series, Band / Volume 43

D 468 (Diss. Wuppertal, Univ., 2020)

ISSN 1868-8489
ISBN 978-3-95806-486-7

Persistent Identifier: urn:nbn:de:0001-2020071402

Vollständig frei verfügbar über das Publikationsportal des Forschungszentrums Jülich (JuSER) unter www.fz-juelich.de/zb/openaccess.

Abstract

Efficient algorithms for the numerical solution of partial differential equations are required to solve problems on an economically viable timescale. In general, this is achieved by adapting the resolution of the discretization to the investigated problem, as well as exploiting hardware specifications. For the latter category, parallelization plays a major role for modern multi-core and multi-node architectures, especially in the context of high-performance computing.

Using finite element methods, solutions are approximated by discretizing the function space of the problem with piecewise polynomials. With *hp*-adaptive methods, the polynomial degrees of these basis functions may vary on locally refined meshes.

We present algorithms and data structures required for generic *hp*-adaptive finite element software applicable for both continuous and discontinuous Galerkin methods on distributed memory systems. Both function space and mesh may be adapted dynamically during the solution process.

We cover details concerning the unique enumeration of degrees of freedom with continuous Galerkin methods, the communication of variable size data, and load balancing. Furthermore, we present strategies to determine the type of adaptation based on error estimation and prediction as well as smoothness estimation via the decay rate of coefficients of Fourier and Legendre series expansions. Both refinement and coarsening are considered.

A reference implementation in the open-source library `deal.II`[1] is provided and applied to the Laplace problem on a domain with a reentrant corner which invokes a singularity. With this example, we demonstrate the benefits of the *hp*-adaptive methods in terms of error convergence and show that our algorithm scales up to 49,152 `MPI` processes.

[1] `https://www.dealii.org`

Zusammenfassung

Für die numerische Lösung partieller Differentialgleichungen sind effiziente Algorithmen erforderlich, um Probleme auf einer wirtschaftlich tragbaren Zeitskala zu lösen. Im Allgemeinen ist dies durch die Anpassung der Diskretisierungsauflösung an das untersuchte Problem sowie durch die Ausnutzung der Hardwarespezifikationen möglich. Für die letztere Kategorie spielt die Parallelisierung eine große Rolle für moderne Mehrkern- und Mehrknotenarchitekturen, insbesondere im Kontext des Hochleistungsrechnens.

Mit Hilfe von Finite-Elemente-Methoden werden Lösungen durch Diskretisierung des assoziierten Funktionsraums mit stückweisen Polynomen approximiert. Bei *hp*-adaptiven Verfahren können die Polynomgrade dieser Basisfunktionen auf lokal verfeinerten Gittern variieren.

In dieser Dissertation werden Algorithmen und Datenstrukturen vorgestellt, die für generische *hp*-adaptive Finite-Elemente-Software benötigt werden und sowohl für kontinuierliche als auch diskontinuierliche Galerkin-Verfahren auf Systemen mit verteiltem Speicher anwendbar sind. Sowohl der Funktionsraum als auch das Gitter können während des Lösungsprozesses dynamisch angepasst werden.

Im Besonderen erläutert werden die eindeutige Nummerierung von Freiheitsgraden mit kontinuierlichen Galerkin-Verfahren, die Kommunikation von Daten variabler Größe und die Lastenverteilung. Außerdem werden Strategien zur Bestimmung des Adaptierungstyps auf der Grundlage von Fehlerschätzungen und -prognosen sowie Glattheitsschätzungen vorgestellt, die über die Zerfallsrate von Koeffizienten aus Reihenentwicklungen nach Fourier und Legendre bestimmt werden. Dabei werden sowohl Verfeinerung als auch Vergröberung berücksichtigt.

Eine Referenzimplementierung erfolgt in der Open-Source-Bibliothek `deal.II`[1] und wird auf das Laplace-Problem auf einem Gebiet mit einer einschneidenden Ecke angewandt, die eine Singularität aufweist. Anhand dieses Beispiels werden die Vorteile der *hp*-adaptiven Methoden hinsichtlich der Fehlerkonvergenz und die Skalierbarkeit der präsentierten Algorithmen auf bis zu 49.152 `MPI`-Prozessen demonstriert.

[1] `https://www.dealii.org`

Acknowledgments

The investigations carried out in this dissertation were performed with the `deal.II` library, and I cannot thank the whole `deal.II` community enough for all their support. In particular, I would like to express my gratitude towards Prof. Dr. Wolfgang Bangerth for hosting me during a seven month research stay at the Colorado State University and for reviewing this thesis, as well as Peter Munch for his interest in both my work and its expansion and for proofreading. I would like to also thank Dr. Denis Davydov and Dr. Martin Kronbichler for their collaboration, as well as Graham Harper and Sean McGovern for lots of `deal.II` related discussions.

I like to emphasize my gratitude to Prof. Dr. Armin Seyfried as the head of my home institute and my supervisor Prof. Dr. Lukas Arnold for making my PhD project possible. I am grateful for their support and encouragement, especially in spending about half a year abroad which allowed me to make huge progress in the project that was mandatory for its completion.

Furthermore, I wish to thank all the people I had the pleasure of meeting and working with during my time at the Forschungszentrum Jülich and the Bergische Universität Wuppertal. Thanks to Dr. Anne Kathrin Küsters and My Linh Würzburger for helpful discussions about numerics and programming and for proofreading, as well as Dr. Alexander Belt for his continuous motivation. In addition, I would also like to mention Dr. Erik Andresen, Anna Braun, Tristan Hehnen, Susanne Hendel, Patrick Lauer, Jette Schumann, Alec Singh, Corinna Trettin, and Ashish Vinayak, who became close friends.

Last, I am grateful to all friends and family for their continuous support on my journey. Be it through hiking adventures, joint sports activities such as running, cycling and swimming, riding roller coasters, attending concerts or simply by spending time together in any other possible way. It is impossible to mention all the people I have in mind here, and I hope that you, as a reader of these lines, feel addressed and associate pleasant memories in the same way as I do. I am grateful to have spent these moments with you and look forward to many more to come. Thank you.

Contents

List of Figures

List of Abbreviations

Chapter 1

Introduction

For the analysis of most problems in science and engineering, mathematical modeling is required to capture underlying correlations and apply findings to related variations. With rising complexity of the model, an analytical solution of a problem so described is less likely to exist, and can only be acquired via approximation with numerical methods. Computers are used today to solve these kinds of problems numerically, but depending on the complexity of the problem and the computer hardware used, such analyses can have exceptionally long execution times.

An intelligent distribution of the computing resources, which due to the dynamics of a simulation does not have to be done *a priori* but progressively, can be used to reduce the computing time (*strong scaling*), and can be used to provide more accurate results in the same execution time (*weak scaling*). This is possible both physically by workload distribution to several processors and logically by an adaptive resolution of the simulation, in each case based on the current state of the simulation. Recent advances in computer technology allow us to solve problems with billions of unknowns. However, raw computing power does not mean we can use it without further ado. Only the combination with algorithms that use all available resources efficiently and scale with the problem size offers a massive potential to reduce execution times. The goal of this dissertation is the provision of such new, efficient algorithms.

Applications can be optimized for the hardware structure down to the processor level, for example using single instruction, multiple data (SIMD) instructions combined with vectorization, and by avoiding bottlenecks caused by memory and network bandwidth. Furthermore, modern multi-core and multi-processor systems require parallelization to make hardware threads and processes cooperate with each other, respectively. Depending on the hardware architecture, many different application programming interfaces (APIs) have been developed over the last decades which allow developers to take opportunity of unified interfaces. For machines with shared memory access like modern desktop workstations, independent computing tasks can be distributed among all hardware threads subject to a work stealing policy, for which Open Multi-Processing (OpenMP)® [1] and Intel® Threading Building Blocks (TBB) [2] are the most prominent approaches. On large-scale supercomputers, processes are spread out on multiple computing nodes with independent memory connected via network. To enable them to cooperate, data needs to be exchanged between all participating nodes. For communication between processes, the Message Passing Interface (MPI) [3] has become a standard. A hybrid combination of both techniques for shared and distributed memory is possible. Recently, streaming multiprocessor architectures on graphics processing units (GPUs) have become of more and more interest for scientific applications, as they offer lots of theoretical throughput, but are strongly limited in flexibility. Open Accelerators (OpenACC)® [4] and Nvidia® Compute Unified Device Architecture (CUDA)® [5] provide interfaces for the scientific use of GPUs.

Most problems from mathematics, nature, and engineering can be classified to be part of continuum mechanics and can be modeled as boundary value problems using partial differential equations. These problems are usually formulated on a subset of the three-dimensional continuous space and can often not be solved analytically, so that we need find a solution with numerical approximations. Numerical methods require the discretization of the continuous space which will be divided into smaller entities that couple with neighboring ones. A large variety of these methods exist, of which we briefly describe the most commonly used ones. With finite difference methods (FDM), differential operators are evaluated as difference quotients on a finite number of grid points. The idea of finite volume methods (FVM) is the preservation of conserved quantities on small volumes by applying the Gauss-Ostrogradski theorem, which results in balancing volumetric averages with fluxes on interfaces of neighboring volumes. In finite element methods (FEM), we specify a function space of piecewise polynomials in which we find the function that satisfies the partial differential equation of the investigated problem as a best approximation. The residual is projected orthogonal to the space of piecewise polynomials, which is equivalent to minimizing the energy for elliptic equations (Brenner and Scott 2008).

In addition to optimizing numerical methods to the hardware, we can also adjust the numerical discretization to the local complexity of the investigated problem by adapting its resolution. This does not only assure the full utilization of all available resources, but also their efficient usage. With adaptive mesh refinement (AMR), or h-adaptive refinement, the spatial resolution of our discretization will be locally assigned, resulting in entities with different sizes or distances h. While FDM requires a regular topology for AMR, it is applicable to FVM and FEM without major restrictions. In addition, FEM offers the unique capability for p-adaptation, in which the polynomial degree of the basis functions is locally set. The combination of both is possible, resulting in hp-adaptive methods.

The methods can be applied on various problems involving partial differential equations from mathematics, nature, and engineering. They have already been extensively used for, e.g., structural mechanics, heat transfer, wave propagation, electrostatics, and fluid dynamics to name just a few. In engineering practice, these methods form the basis for simulations on objects, for instance to investigate their stress and wear behavior and to generate their flow profile. Corresponding model experiments are complex and expensive, so computer simulations offer an alternative tool in engineering applications. A concrete application example describes the simulation of smoke spread in buildings. On the basis of their results, fire safety engineers are able to optimize smoke extraction systems and evacuation routes to increase the safety of civilians in the event of a fire outbreak. In general, fires remain spatially localized even after their ignition phase, so the dynamic allocation of both resolution and computational resources is highly favorable in this scenario. Their simulation on large scale buildings or connected facilities like underground tunnel systems as investigated in the ORPHEUS project (Arnold 2017), yield an incredible workload.

Thus, the combination of parallelization and adaptive methods is necessary to perform simulations on an economically acceptable time scale. However, their implementation is very difficult with many technical finesses to consider. Many software solutions for parallel h-adaptive methods exist, however their hp-adaptive equivalents are rarely realized because of their complexity. In this dissertation, we will focus on hp-adaptive FEM with their exceptional error convergence properties (Guo and Babuška 1986; Babuška and Guo 1996), and provide their parallelization for distributed memory systems.

In the past, several algorithms for parallel hp-adaptive FEM have been developed, but they always stayed in the context of discontinuous Galerkin (DG) methods which allow solutions to be discontinuous across cell interfaces. For example for Navier-Stokes problems, Paszyński and Demkowicz (2006) and Chalmers et al. (2019) presented methods for distributed memory architectures, while Paszyński and Pardo (2011) and Jomo et al. (2017) demonstrated methods for shared memory machines. A general approach which also works with continuous Galerkin (CG) methods poses additional implementation challenges that are pointed out in the course of this dissertation.

Due to their complexity, *hp*-adaptive methods have always stayed in an experimental stage and have never been prepared to be easily applied by a broader academic audience, especially in combination with parallelization. Though, there are several open-source libraries available to the public that provide the bare functionality for *hp*-adaptive FEM on distributed memory architectures using the MPI protocol, such as the libraries `PHAML` (Mitchell 2002; [6]), `PHG` (Zhang 2019; [7]), and `MoFEM` (Kaczmarczyk et al. 2020a; [8]). However, even here the application of these features is not immediately accessible to the end user. We are not aware of any commercial tools capable of this feature.

Furthermore, although applications of parallel *hp*-adaptive FEM have been presented thoroughly, there is no systematic description on how to realize them yet as a software implementation. Algorithms and data structures have already been presented in detail for parallel *h*-adaptive FEM by Bangerth, Burstedde, et al. (2012) and sequential *hp*-adaptive FEM by Bangerth and Kayser-Herold (2009). The goal of this dissertation is to provide the combination of both algorithms highlighting difficulties to combine both parallelization with *hp*-adaptive methods. This dissertation is not meant to be an in-depth guide for the creation of FEM software. We would rather like to emphasize the basic ideas for parallel *hp*-adaptive FEM and point out programming challenges. We will provide an example implementation in the `deal.II` library (Bangerth, Hartmann, and Kanschat 2007; [9]), so that the reader is able to either embed our findings into their own FEM code or use the `deal.II` implementation right away.

`deal.II` is an open-source software library for the creation of general purpose FEM codes. It is part of the extreme-scale scientific software development kit (xSDK) (Bartlett et al. 2017; [10]), which combines efforts of many research software engineers to make their expertise in optimized high-performance computing (HPC) available and provides them to the public as a whole. In this context of sharing knowledge, `deal.II` profits from parallel linear algebra provided by the open-source libraries `Trilinos` (Heroux et al. 2005; [11]) and `PETSc` (Balay et al. 2019; [12]), and utilizes their implementation via designated interfaces.

Furthermore, `deal.II` does not provide the hierarchic generation of *h*-adaptive meshes and their partitioning on multiple processes of distributed memory architectures itself, but relies on the implementation of the open-source library `p4est` (Burstedde, Wilcox, and Ghattas 2011; [13]). In this regard, `p4est` is used as an 'oracle': Operations that manipulate the mesh and its partitioning happen on a distributed structure provided by `p4est`, and will be recreated only on the locally owned sections of the `deal.II` mesh on every process with a set of queries to the master mesh.

Since the ideas of this dissertation will be realized in `deal.II`, we capitalize on their hierarchic quadtree and octree structures with corresponding quadrilateral and hexahedral cells in two and three dimensions, which are arranged by means of a Z-order or Morton space-filling curve. However, the presented ideas for algorithms and data structures in this dissertation are not restricted to these specifications.

In this dissertation, we present all enhancements necessary to supply parallel *hp*-adaptive methods for algorithms and data structures that are already capable of parallel *h*-adaptive and sequential *hp*-adaptive FEM. In Ch. 2 we present the necessary details for static meshes, i.e., meshes with fixed resolution and fixed assignment of finite elements from beginning to end of a simulation. Ch. 3 deals with all necessities of dynamic *hp*-adaptive methods, and presents algorithms to automatically determine regions to adapt. We apply the presented methods on a simple numerical example in Ch. 4 to show the benefits of *hp*-adaptive methods and scalability on the JURECA supercomputer (Krause and Thörnig 2018; [14]).

All algorithms presented in this dissertation are available starting with release 9.2.0 of the `deal.II` library (Arndt et al. 2020; [9]), with which all numerical examples in this dissertation were performed.

Chapter 2

Static parallel *hp*-adaptive finite element methods

Any kind of numerical method requires thorough thought on designing suitable algorithms and data structures with respect to correctness, robustness and performance. In general, the ideas behind implementations of these methods are similar and can be generalized. This is also the case for additional enhancements that improve basic realizations. In the case of the finite element method (FEM) such features are, for example, parallelization, adaptive methods, continuous or discontinuous Galerkin methods, and the support for complex geometries.

In this chapter, we will present algorithms and data structures for parallel, *hp*-adaptive finite element methods. For now, we focus on static meshes with a fixed grid resolution and assignment of finite elements that will not change during the course of the application. Those can be used for multi-physics applications, in which different characteristics can be assigned to certain parts of the domain using dedicated finite elements that couple with each other, for example, a Stokes fluid interacting with an elastic solid [15].

Generalized thoughts on the following two aspects have already been presented: Bangerth and Kayser-Herold (2009) developed a general implementation for *hp*-adaptive FEM software; and a generalized distributed computing approach of the finite element method has been introduced by Bangerth, Burstedde, et al. (2012). However, the consolidation of both features is not trivial, thus this chapter should be understood as an enhancement of the two aforementioned contributions and bases in large parts on it. For details, we recommend a previous reading of both articles, since only relevant key aspects are revised in the following.

We will elaborate on the nontrivial aspects of parallel, *hp*-adaptive FEM in the following sections, after providing a brief overview about the basics that are necessary for their understanding. These nontrivial aspects are the unique enumeration of degrees of freedom (DoFs) in the context of continuous Galerkin (CG) methods, as well as load balancing. All features added to the `deal.II` library in the context of *hp*-adaptive FEM can be found in a development log provided in [16].

2.1 Prerequisites

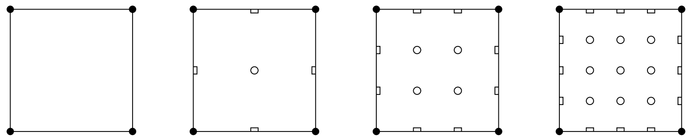
The basic idea of the finite element method (FEM) consists of the specification of a finite-dimensional function space and finding a solution to the investigated boundary value problem in it. To be more precise, we pick a suitable set of basis functions for which the solution is a linear combination. Its coefficients are called unknowns or degrees of freedom (DoFs), since their values are determined after solving the problem.

Differential equations are solved on prescribed domains Ω, which are typically part of $\mathbb{R}^2$ or $\mathbb{R}^3$, and form boundary value problems with boundary conditions posed on $\partial\Omega$. Their numerical solution with FEM requires the subdivision of the domain into a triangulation of smaller cells K, where $\Omega = \bigcup_i K_i$ and each cell K has a nonempty interior and a Lipschitz-continuous boundary. Typical choices for cells are triangles or quadrilaterals in two and tetrahedra or hexahedra in three dimensions. Further, on each cell K, we define a finite-dimensional function space P, accompanied by a set of functionals $N = \Psi_1, \ldots, \Psi_n$ that are a basis for its dual space P' . Ciarlet (1978, Sec. 2.1) defined such tuples (K, P, N) as finite elements.

With this definition, we are able to find a set of shape functions $\varphi_1, \ldots, \varphi_n$ which forms the basis for the finite element space and characterizes its dimension. Consequently, a finite element approximation is a linear combination of these functions $u_{\mathrm{hp}}(\boldsymbol{x}) = \sum_j U_j \, \varphi_j(\boldsymbol{x})$, where coefficients U_j are known as degrees of freedom.

We transform the original boundary value problem into a variation problem by converting it to a weak formulation. This process belongs to the class of Galerkin methods, of which we distinguish between continuous Galerkin (CG) and discontinuous Galerkin (DG) methods. For the former, we require that our finite element approximation is continuous across cell interfaces resulting in shared DoFs on all transitions. For discontinuous methods, jumps of the approximation are possible on cell interfaces, and thus each cell has its own set of DoFs of which none are shared. Here, relations between cells are quantified via penalty methods that correlate DoFs on interfaces of neighboring cells.

With the discretized weak formulation of the investigated problem, we are



(a) Q_1 *element.* (b) Q_2 *element.* (c) Q_3 *element.* (d) Q_4 *element.*

Figure 2.1: *First four Lagrange elements Q_p on quadrilateral unit cells with $(1+p)^{\dim}$ support points each. Here, we distinguish between support points on vertices (•), lines (□) and quadrilaterals (○).*

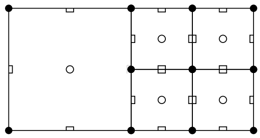
left to assemble and solve the corresponding system of linear equations for the unknown coefficients or DoFs. In practice, we calculate the appearing integrals using quadrature rules and basis functions which have been mapped from a finite element on a reference cell $\widehat{K}$. It is thus sufficient to define reference finite elements and assign them to cells.

One way to choose finite elements is to use nodal functionals Ψ_i which evaluate the values of a function φ at their associated support points $\boldsymbol{x}_i$, i.e., $\Psi_i[\varphi] = \varphi(\boldsymbol{x}_i)$. The corresponding shape functions form a basis of P dual to N fulfilling $\Psi_i[\varphi_j] = \delta_{ij}$, where δ is the Kronecker delta function. In other words, shape functions are designed to have the value one on their associated support point and the value zero on all others. When dealing with this category of finite elements in practice, it is sufficient to work with support points and shape functions once the type of element is specified. One of the most common types of finite elements are Lagrange elements Q_p, that are based on Lagrange interpolation with polynomials of order p and can be extended from one to higher dimensions via tensor products. The arrangement of support points on quadrilaterals in Lagrange elements up to fourth order polynomials are depicted in Fig. 2.1.

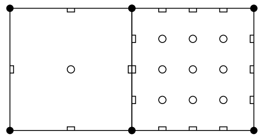
This is just a brief introduction to FEM in order to provide the fundamentals for this chapter and to specify the nomenclature used. More details on this topic can be found in common literature (e.g., Quarteroni and Valli 1994; Ern and Guermond 2004; Elman, Silvester, and Wathen 2014). Especially Brenner and Scott (2008) provided a more rigorous and mathematically sound definition of finite elements, on which this summary was based.

2.1.1 Adaptive methods for FEM

In computational applications, adaptive methods are used to align the resolution or rather the computational resources to the complexity of the investigated problem, i.e., to specific parts of the domain. In terms of grid refinement on quadrilateral and hexahedral meshes, cells will be split once per dimension,



(a) *h-adapted mesh with Q_2 elements. The smaller cells on the right half are the result of isotropic refinement.*



(b) *p-adapted mesh with one Q_2 element on the left and one Q_4 element on the right half.*

Figure 2.2: *Example of differently adapted meshes consisting of initially two quadrilateral cells, from which the right one has been refined. Differences in the refinement level of neighboring cells give rise to hanging nodes. Note that both scenarios yield the same number of support points.*

resulting in four or eight children in two or three dimensions, respectively. Similarly, we join the corresponding amount of cells to their parent cell for grid coarsening. This process is also known as h-adaptation, referring to adjusting the cell's length or diameter h locally. We require hierarchic relations between parent cells and their children that motivate the use of tree structures rooting in each cell of the initial coarse mesh. Combined, this leads to a forest structure. The entirety of all leaves in this forest corresponds to a full representation of the domain which is required for assembling and solving the system of equations. We call cells without children *active* cells, since they are the only ones carrying DoFs. For more information, Bangerth and Rannacher (2003) elaborate more rigorously on grid adaptation, especially for the FEM method.

As an alternative to modifying the grid resolution, we can also adapt the function space using various finite elements associated with each cell. These finite elements may differ in the polynomial degree p of their shape functions, offering the unique possibility for p-adaptation, or hp-adaptation when used together with grid adaptation. In practice, we specify a collection of reference finite elements, one of which is assigned to each cell in the domain. We identify the currently assigned finite element with an *active finite element index* on each cell.

Refinement level differences on quadrilateral or hexahedral meshes, as well as varying finite elements on neighboring active cells lead to *hanging nodes* which are nodes with no counterpart on the opposite side of the interface. A depiction of h- and p-adapted domains along with hanging nodes is presented in Fig. 2.2. In combination with CG methods, the finite element approximation needs to be continuous on these hanging nodes, which requires constraining them to the surrounding ones on the shared interface.

For h-adaptive methods, we need to constrain hanging nodes from cells of a finer refinement level to the neighboring coarser cell by interpolation. A typical situation is depicted in Fig. 2.2a. For convenience, we limit the level difference of neighboring cells to one level. For quadrilateral or hexahedral meshes, this 2:1 mesh balance ensures to limit the occurrence of hanging nodes.

Whenever different finite elements face each other in case of p-adaptation, we need to impose continuity on these interfaces as well. This is performed by restricting the element with the higher polynomial degree to the continuity of the lower one. Following the example from Fig. 2.2b, we would have to restrict the additional nodes of the Q_4 element to the ones of the Q_2 element on their shared line. To formulate it in a more general way, we need to constrain continuity of all neighboring finite elements to their smallest common finite element subspace. We say that this particular element *dominates* the others.

There may be cases in which neighboring elements do not dominate each other, since they do not pose compatible continuity requirements. As an example, this would be the case for two neighboring vector-valued elements $(Q_2 \times Q_1)$ and $(Q_1 \times Q_2)$. In this case, we designate the first active finite element on a mesh object to be dominating.

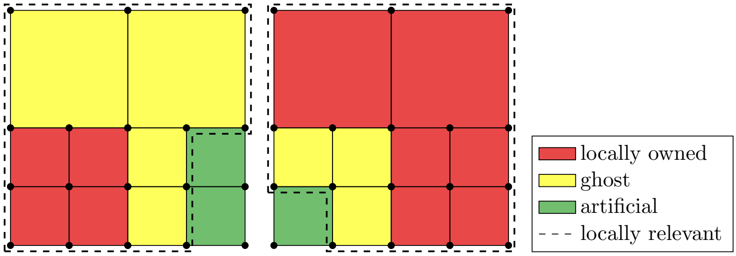
This section summarized data structures and algorithms for hp-adaptive FEM presented in great detail by Bangerth and Kayser-Herold (2009).

2.1.2 Parallelization of FEM

High-performance computing (HPC) applications require scalable algorithms and data structures for machines with distributed memory, which we realize using the Message Passing Interface (MPI) standard [3]. In this context, the workload on all participating processes will be shared by partitioning the mesh among them.

This approach poses different challenges on the design of data structures and information exchange between MPI processes. Especially dynamic changes of the domain by hp-adaptation are difficult, since they require the redistribution of the workload by repartitioning the global mesh, as well as the transfer of data. In addition, the workload is no longer easily predictable by a simple measure such as the number of cells each process owns. We will discuss this topic in detail later in Ch. 3. For now, we distinguish between different subsets of cells and DoFs which will be introduced in the following. An example of a distributed, adapted mesh is shown in Fig. 2.3 with all presented types of cells.

Each process will only store data of cells that belongs to its owned section of the domain, which we call *subdomain*. We call these cells *locally owned cells*, and all DoFs on them are referred to as *locally active DoFs*. Every DoF will be uniquely enumerated on the global mesh. With CG methods, DoFs on interfaces between cells of different processes may be owned by either one or



(a) *Process with rank 0.* (b) *Process with rank 1.*

Figure 2.3: *Exemplary partitioning of a parallel distributed h-adapted mesh on two MPI processes. Every cell is owned by exactly one process. Properties of cells on their respective process are highlighted.*

the other process. Thus, not all active DoFs on locally owned cells have to be *locally owned DoFs*.

During the assembly of equation systems, we need to refer to surrounding cells that do not belong to the local domain. A requirement for the parallelization of FEM is the provision of data on them via communication. Typically, a halo spanning one level of cells around the locally owned domain covers the necessary cells which we call *ghost cells*. Data from ghost cells has to be requested from the owning process. The combination of locally owned and ghost cells and their corresponding DoFs is labeled *locally relevant*.

Further, we require that every process stores a copy of the common coarse mesh. This allows a straightforward construction of the whole grid during repartitioning just with adaptive methods. We establish the 2:1 mesh balance on all cells of the local mesh, leaving refined cells on regions that are not locally relevant. These cells that are not locally relevant will be called *artificial cells*.

Heister (2011) and Bangerth, Burstedde, et al. (2012) described the parallelization of h-adaptive FEM within the `deal.II` library [9] in great detail.

2.2 Enumeration of degrees of freedom

Formulating and solving the system of linear equations requires an unique identification of all involved DoFs in the global mesh.

DoFs are associated with mesh objects, i.e., vertices, edges, faces, and cells. If support points are located on interfaces between neighboring cells in the context of DG methods, they are assigned separate DoFs on each cell. Thus, the enumeration of both DoFs and cells can happen analogously. However, CG methods require that shared DoFs on interfaces between neighboring cells are unique. Thus, each DoF has to relate to a single cell, or in other words, will be owned by a single cell. This assignment is crucial for the efficient preparation of relevant distributed data structures.

The latter scenario poses challenges in the enumeration of DoFs when considering either parallelization or *hp*-adaptive methods, let alone a combination of both. A first attempt to cope with this problem would involve so called constraints: We enumerate all DoFs on each cell independently, but identify two similar DoFs as identical during the assembly of the equation system. Although this approach would be easy to implement, we would needlessly add DoF duplicates to the equation system, sacrificing performance by wasting memory and computing time. We conclude that a unique enumeration of DoFs is mandatory for a robust FEM implementation for CG methods.

Both Bangerth, Burstedde, et al. (2012) and Bangerth and Kayser-Herold (2009) have dealt with DoF enumeration with parallelization and *hp*-adaptive methods, respectively, and presented algorithms for each case, but the combination of both is not trivial. In this section, we will briefly summarize each implementation and present an enhanced algorithm in detail for the unique identification of DoFs for CG methods with parallel *hp*-adaptive methods.

In the following, all algorithms will be presented independently of the underlying data structures and should be easily applicable to any kind of existing FEM code. Results will be presented with our reference implementation in the `deal.II` library.

For *hp*-adaptive FEM, the algorithm proposed by Bangerth and Kayser-Herold (2009, Sec. 4.2) enumerates all DoFs on each cell consecutively in a first step, and then unifies these shared DoFs on cell interfaces by favoring the index of the dominating finite element.

In parallel applications, the enumeration of DoFs on interfaces between neighboring subdomains pose a problem: They have to be assigned to one of them, for which Bangerth, Burstedde, et al. (2012, Sec. 3.1) proposed to use a certain *tie-break* criterion as a decision aid. Their algorithm starts with enumerating DoFs on all locally owned cells. On interfaces between subdomains, all DoFs will be assigned to the process with the lower rank and thus keep the index from the subdomain with the lower identifier. Once ownership of all DoFs is clarified, their index will be increased by the number of DoFs owned by processes with a lower rank. Now, every locally owned DoF has its final index assigned. Each process needs to know all locally relevant DoFs for the solution of the equation system, which requires exchanging DoF indices on ghost cells via point-to-point communication. This operation has to be performed twice since there may have been DoFs on ghost cells of which the owning process did not know the correct indices of after the first exchange.

For parallel *hp*-adaptive FEM, the mere concatenation of both algorithms

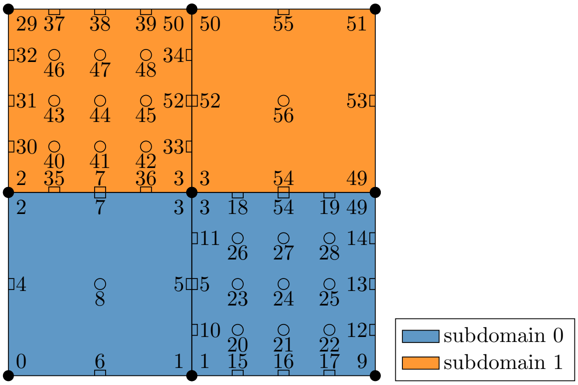


Figure 2.4: *Benchmark scenario to verify our algorithm for DoF enumeration. The mesh is distributed on two MPI processes, each owning one Q_2 and one Q_4 element. All DoFs are uniquely identified on the global mesh as a result of the enumeration algorithm from Sec. 2.2.*

does not suffice. The case in which distinct finite element types from different subdomains are adjacent has to be given special consideration. We could cope with this situation by constraining DoFs on these interfaces to each other. However, this would leave unnecessary DoF duplicates in the equation system. Additionally, the global number of DoFs would differ with the number of subdomains in this approach. We would rather keep it independent from the number of processes, which would simplify debugging and assures that our solvers yield the same results on any number of subdomains. The algorithm developed in this thesis meets this requirement and combines the ideas of both previous algorithms.

A suitable benchmark that we used to test the enumeration algorithm consists of a two-dimensional mesh of four neighboring cells. We provide two different Lagrangian elements that share at least one additional DoF per cell interface than only on vertices. For this purpose, we choose Lagrangian elements Q_2 and Q_4. Each of these finite elements will be assigned to two catty-cornered cells. Furthermore, we divide the mesh into two subdomains containing two neighboring cells with different reference finite elements. The whole setup is shown in Fig. 2.4 and covers all combinations of adjacent finite elements that we have encountered so far in the parallel *hp*-adaptive context. A step-by-step demonstration of the algorithm on this particular benchmark is presented in App. A.

The algorithm enumerates all DoFs on locally relevant cells, which includes ghost cells. Thus, we begin by exchanging active finite element indices on ghost cells so that we have information about all locally relevant finite elements and can prepare all data structures accordingly. We do this with point-to-point communication.

In short, our algorithm is based on the parallel algorithm by Bangerth, Burstedde, et al. (2012, Sec. 3.1) for the most part. We will add a DoF unification step after enumerating all locally owned cells, while subjecting to the finite element domination hierarchy to decide ownership on all interfaces. After exchanging DoFs on all ghost cells, we are left to merge the valid DoFs on interfaces with the valid counterparts.

In detail, the complete algorithm consists of the following six steps, starting with an initialization step flagged with '0'. The step-by-step demonstration presented in App. A should be consulted as an aid to its understanding. To follow the same nomenclature as Bangerth, Burstedde, et al. (2012), we call the set of all locally owned cells $\mathbb{T}^p_{\text{loc}}$, the set of all ghost cells $\mathbb{T}^p_{\text{ghost}}$, and the set of all locally relevant cells $\mathbb{T}^p_{\text{rel}} = \mathbb{T}^p_{\text{loc}} \cup \mathbb{T}^p_{\text{ghost}}$ on processes and subdomains identified by the integer p.

0. *Initialization.* On all locally relevant cells $K \in \mathbb{T}^p_{\text{rel}}$, DoF indices are set to an invalid value, for example $i := -1$.

1. *Local enumeration.* Iterate over all locally owned cells $K \in \mathbb{T}^p_{\text{loc}}$ and assign valid DoF indices in ascending order, starting from zero.

2. *Tie-break.* Go over all locally owned cells $K \in \mathbb{T}^p_{\text{loc}}$. If a mesh object on K is also part of an adjacent ghost cell which has the same reference finite element assigned and belongs to a subdomain of lower rank $q < p$, then invalidate all DoFs on the mesh object by setting their index to the invalid value i.

3. *Unification.* Go over all locally owned cells $K \in \mathbb{T}^p_{\text{loc}}$. For all shared DoFs on interfaces to neighboring cells, constrain the DoF of the dominated finite element to the one of the dominating element. On interfaces to ghost cells, set DoFs indices to the invalid value i if the dominating element is assigned to the ghost cell. It is possible that none of the adjacent elements dominates, e.g. if a $(Q_1 \times Q_2)$ element neighbors a $(Q_2 \times Q_1)$ element as described in Sec. 2.1. In this case, designate the first active finite element on each mesh object as its dominant one. Populate a list with pairs of identifiers for these constrained DoFs.

At this stage, each process knows which DoFs are owned by them.

4. *Global re-enumeration.* Iterate over all locally owned cells $K \in \mathbb{T}^p_{\text{loc}}$ and enumerate those DoF indices in ascending order that have a valid

value assigned, while considering all constraints from the previous step. Store the number of all valid DoF indices on this subdomain as n_p. In a next step, shift all indices by the number of DoFs that are owned by all processors of lower rank $q < p$, or in other words, by $\sum_{q=0}^{p-1} n_q$. This corresponds to a prefix sum or exclusive scan, and can be obtained via `MPI_Exscan` [3]. Note that simply shifting indices as in the algorithm without p-adaptivity is no longer sufficient.

At this stage, all subdomains and processes have the correct indices assigned to all locally owned DoFs, which are enumerated consecutively.

5. *Ghost exchange.* Communicate DoF indices from locally owned cells $K \in \mathbb{T}^p_{\text{loc}}$ to other processes using point-to-point communication as follows.

   a. Prepare containers with data to be sent from subdomain p to each adjacent subdomain q.

   b. Loop over all locally owned cells that have ghost neighbors. Add the cell identifier with all associated DoF indices to every data container that corresponds to an adjacent subdomain of the current cell.

   c. Send each data container to its designated destination process with nonblocking point-to-point communication via `MPI_Isend` [3].

   d. Receive data containers from processes of adjacent subdomains with nonblocking point-to-point communication via `MPI_Irecv` [3]. The received data corresponds to all ghost cells of this subdomain p. On each of these cells, set the received DoF indices accordingly.

   All communication in this step is symmetric, which means that a process only receives data from another process when it also sends data to it. Thus, there is no need to negotiate communication.

After the previous ghost exchange each DoF on interfaces with ghost cells has exactly one valid index assigned.

6. *Merge on interfaces.* Go over all locally relevant cells $K \in \mathbb{T}^p_{\text{rel}}$. On interfaces between locally owned and ghost cells, set all remaining invalid DoF indices with the corresponding valid one of the dominating finite element.

At this stage, all locally owned cells $K \in \mathbb{T}^p_{\text{loc}}$ have their correct DoF indices set.

7. *Ghost exchange.* Some ghost cells may still have invalid DoF indices assigned since the owning process did not know all correct indices on this

particular cell when the last communication happened. Another ghost exchange closes this gap by repeating step 5. However this time, only data from those cells has to be communicated which had invalid DoF indices prior to step 5d.

One variant of this algorithm would be to modify steps 2 and 3 inasmuch as we apply the *tie-break* criterion on all DoFs on ghost interfaces and exclude them from DoF unification entirely. However, this would not assign shared DoFs to the dominating finite element on ghost interfaces, which would be inconsistent compared to the locally owned parts of the domain.

At the end of this algorithm, all global DoF indices have been set correctly, and every process knows the indices of all locally relevant DoFs: All DoFs on interfaces belong to the dominating finite element on the process with the smallest rank. In particular, while the algorithm is substantially more complicated than the one without p-adaptivity, no additional communication steps are introduced to the two original ones from the h-adaptive variant.

There may be situations where DoFs are constrained to others, which in turn are constrained to different third ones. These chains of constraints may span over DoFs from multiple subdomains. To deal with this case, we might need more than the current two communication steps in our algorithm. However, we could not think of any scenario in which this is going to be the case, and did not encounter any issues in our investigations so far.

In three-dimensional scenarios, Bangerth and Kayser-Herold (2009, Sec. 4.6) pointed out possible complications with circular constraints during DoF unification whenever three or more different finite elements share a common edge. We still have not figured out a satisfactory solution for this problem and conform to their original way to cope with this situation: All DoFs on these edges will be excluded from the unification step and will be treated separately via constraints.

2.3 Load balancing

The efficient use of all computational resources requires a uniform distribution of all workload among them. There are many factors that determine the workload in a FEM application, above all the preparation of data structures, the assembly of both the matrix and right hand side of the linear equation system, and the choice of the type of solver. Each of these categories contributes a different amount to the total workload, with the solver accounting for the largest share in general. We should thus balance the number of rows or nonzero entries per process in the solution matrix. However, information about the matrix is available late in a solution cycle, so we need to look for a different measure at an earlier stage.

In most h-adaptive applications, cells are similar and correspond to the same workload. Thus, we can simply balance the number of cells on all processes. However with hp-adaptive applications, this is no longer the case due to the diversity of finite elements. In this case, since the domain is partitioned on the basis of cells, we need to assign a corresponding weight to every cell that determines its individual workload and balance the cumulated weights among all processes.

The workload of each cell depends on its assigned reference finite element and quadrature formula, and correlates to the number of DoFs and quadrature points, among other quantifiable values that depend on the individual problem. For example, Lagrangian elements of different order as depicted in Fig. 2.1 each have a distinct number of DoFs.

For the purpose of load balancing, Burstedde, Wilcox, and Ghattas (2011, Sec. 3.3) provided an algorithm for weighted partitioning and enhanced `p4est` [13] with a corresponding implementation, of which we take advantage in `deal.II`. Omitting details about the communication between processes, we will briefly outline its basic idea: On a distributed mesh, calculate the prefix sum of cell weights in the global scope, determine the partition boundaries with a binary search, and transfer cells to their new owning processes if necessary.

In the context of hp-adaptive FEM applications, we identify the assembly of the linear equation system and its solution as the most expensive tasks, and correlate their individual contribution to the workload on each cell with its number of DoFs provided by the associated finite element. The total workload of iterative solvers combined with multigrid preconditioning ideally scales with the global number of DoFs, i.e., $\mathcal{O}\left(N_{\text{dofs}}\right)$. We therefore expect that the corresponding workload of each cell attributed to the solution process also scales with the number of cell-related DoFs, i.e., $\mathcal{O}\left(n_{\text{dofs}}\right)$. Furthermore, we suppose that the workload of each cell for the matrix assembly will be of order $\mathcal{O}\left(n_{\text{dofs}}^2\right)$ since all DoFs in a cell couple with each other.

We expect that the actual workload per cell of an hp-adaptive FEM application will actually scale with an order somewhere in between the two, i.e., $\mathcal{O}\left(n_{\text{dofs}}^c\right)$ with a constant exponent $c \in [1, 2]$. We use this strategy for investigations in Ch. 4 in which we also determine a suitable exponent c.

Independent of the approach just described, weighting each cell with a linear combination $(a\, n_{\text{dofs}}^2 + b\, n_{\text{dofs}})$ also appears conceivable, for which the partitioning results depend on the ratio of both constants a and b, rather than an exponent c.

A reliable measure of weights is tied to the type of problem and the choice of the solver. With the presented approach, we still have to specify a suitable weight manually. We are not aware of any procedure to automatically determine them.

Chapter 3

Dynamic parallel *hp*-adaptive finite element methods

Up to this point, we have only dealt with static *hp*-adaptive meshes, in which the grid resolution and the distribution of finite elements is prescribed from the beginning. However, the key feature of *hp*-adaptive methods is the dynamic distribution of these attributes. The goal is to reduce the overall error of the approximation on the basis of the complexity of the current state of the solution. With dynamic methods, meshes can be tailored to a specific problem iteratively with so called SOLVE-ESTIMATE-MARK-REFINE cycles: After solving the problem on a coarse mesh, it will be adapted based on an error estimate several times, as we will demonstrate in Ch. 4. Also, time dependent problems benefit from dynamic adaptation, in which mesh attributes will be adjusted on the basis of the solution that evolves in time, for example, for heat transfer [17] and fluid flow problems [18].

In practice, we distinguish between two ways of imposing these *hp*-adaptive methods: either manually or automatically. The combination of both *h*- and *p*-adaptation poses new challenges. We will present these issues in this chapter along with algorithms to automatically determine which regions to adapt and which type of adaptation to impose.

First, considerations regarding the implementation of *hp*-adaptivity will be presented. We then propose methods to automatically decide which cells to adapt and how by providing corresponding adaptation and decision strategies. The presented strategies are intended for general purposes and are based on error estimation, error prediction and smoothness estimation. It is conceivable to tailor strategies around particular observables of individual problems, which is not subject of this dissertation. Furthermore, dynamic changes on the mesh require data exchange between processes, which we describe in the end.

New features for *hp*-adaptation as well as a generalized interface have been introduced in the `deal.II` library [9] during working on this dissertation. A development log on all features introduced in the context of dynamic *hp*-adaptive finite element methods (FEM) can be found in [19].

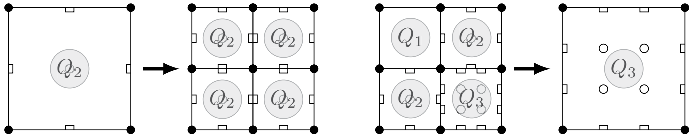
3.1 Realization of *hp*-adaptation

For *hp*-adaptive methods, we need to find ways to prescribe which cells are subject to which kind of adaptation. This grants awareness on how the mesh will change if adaptation is executed.

To indicate that adaptation is about to happen, we introduce general flags for refinement and coarsening into our implementation, respectively. Furthermore, to indicate that *p*-adaptation is going to happen, we specify so called *future finite element indices* that determine the reference finite element from the collection that will be associated to that particular cell after adaptation has been performed. They act as a counterpart to the previously introduced *active finite element indices* in Sec. 2.1. In total, we thus have three different indicators for adaptation: flags for refinement and coarsening, as well as future finite element indices.

To determine the extent to which cells change during the adaptation process, the affected cell properties have to be hierarchically ordered. While for *h*-adaptive methods, a grid hierarchy naturally evolves from the underlying tree or forest data structure, this is not necessarily the case for *p*-adaptation. Here, a hierarchy needs to be provided for the collection of reference finite elements, so that we can assign superior and subordinate elements in case of *p*-refinement or *p*-coarsening, respectively. For example, we arrange Lagrangian finite elements by their polynomial degree in ascending order with the largest degree on the highest level.

Executing *h*-adaptation on a *p*-adapted mesh reveals another challenge. We need to find a suitable finite element on cells that will be *h*-adapted. During *h*-refinement, we can easily choose the finite element from the parent cell for all of its children, but for *h*-coarsening, the choice is not trivial. From all cells that are going to be *h*-coarsened, we pick the one finite element for the parent cell from those assigned to its children that encapsulates all of them. With simultaneous *h*- and *p*-adaptation, future finite elements will be considered instead of the active ones. This conforms to the finite element domination logic that has been introduced by Bangerth and Kayser-Herold (2009) and is described in Sec. 2.1. An example on how finite elements are distributed during *hp*-adaptation is shown in Fig. 3.1.



(a) *h-refinement.* (b) *h-coarsening.*

Figure 3.1: *Inheritance of cell characteristics through h-adaptation in the context of hp-adaptive meshes. With h-refinement, all children will be associated with the parent finite element, while during h-coarsening, the finite element space chosen on the parent cell encapsulates all those of its children* ($Q_1 \subset Q_2 \subset Q_3$).

In a typical adaptation workflow, we have to distinguish between h- and p-adaptation. In our implementation approach, we make a complete distinction between the two at a certain point. First, corresponding cells will be either marked for refinement or coarsening and will be assigned with a corresponding general flag. On all flagged cells, we decide whether to impose p-adaptation by setting a future finite element index or not. Now, meanings of all refinement and coarsening flags change from the general indication of adaptation to signal h-adaptation only. If a cell shall be p-adapted only, the corresponding flags need to be removed and only the future finite element index has to prevail, otherwise both h- and p-adaptation will happen simultaneously. This approach offers full flexibility to let either users decide manually how to adapt, but also provides a sufficient interface for an automatic specification of these mesh attributes, which will be described in the following section.

3.2 Decision criteria

With the bare functionality of dynamic hp-adaptive methods at hand, we need to find ways to automatically decide which parts of the domain we want to adapt and how. For this purpose, we recall the error convergence properties of FEM in this section.

A common observation is that increasing the grid resolution or the polynomial degree of the basis functions will decrease the difference between the finite element approximation u_{hp} and the actual solution u.

In fact, the impact of these adaptation techniques on the error is well understood. Babuška and Suri (1990, Thm. 3.4) determined an upper bound for the error that depends both on the cell diameter h and the polynomial degree p:

$$\|e_{\mathrm{hp}}\|_{H^1(\Omega)} \leq C\, h^{\mu}\, p^{-(m-1)}\, \|u\|_{H^m(\Omega)}\,, \tag{3.1}$$

where $e_{\mathrm{hp}} = u - u_{\mathrm{hp}}$ denotes the error function, m is a measure for the regularity of the solution u, C is a constant dependent on m, and $\mu = \min(p, m-1)$.

These modifications do not have to happen uniformly on a global scale, but can be applied locally, since the global error consists of the local contributions of each cell K:

$$\|e_{\mathrm{hp}}\|^2_{H^1(\Omega)} = \sum_{K \in \Omega} \|e_{\mathrm{hp}}\|^2_{H^1(K)} \, . \tag{3.2}$$

Thus it all comes down to finding those regions that have a significant contribution to the global error, and mitigate their impact by local adaptation.

On suitably chosen *hp*-adaptive meshes, Guo and Babuška (1986, Thm. 5.1) and Babuška and Guo (1996, Thm. 2.5.2, Thm. 3.5.1) even predicted exponential convergence for elliptic problems under the assumption that the solution u is sufficiently regular:

$$\|e_{\mathrm{hp}}\|_{H^1(\Omega)} \leq C \, \exp\left(-b \, N_{\mathrm{dofs}}^{\alpha}\right) \, , \tag{3.3}$$

where constants $b > 0$ and C are both independent of the total number of degrees of freedom (DoFs) N_{dofs}, and $\alpha = 1/3$ for two or $\alpha = 1/5$ for three dimensional problems.

With sufficient information about the investigated scenario, an *hp*-adaptive grid can be tailored to its specifics manually. However, grid adjustments by hand may not be optimal. Furthermore, not all peculiarities about the scenario are generally known in advance, which is especially the case for problems with complex geometries and time dependent problems.

Hence we need to elaborate on algorithms to automatically decide which subsets of the domain qualify for adaptation. With this technique, we typically set up a coarse mesh along with basis functions of a low polynomial degree and obtain a tailored mesh after several adaptation iterations.

In this section, we present different ways to identify areas whose adaptation will be most profitable, and to choose the most beneficial type of adaptation. For *hp*-adaptation in particular, Mitchell and McClain (2014) reviewed and compared a selection of strategies in detail. We demonstrate a subset of their recommendations in terms of performance and applicability, i.e., those strategies that only require the locally relevant part of the current solution.

3.2.1 Adaptation strategies

We will decide on the basis of adaptation criteria on each individual cell whether it will be considered for adaptation. Typical criteria involve comparing errors or their estimates to some absolute or relative threshold. Alternatively, also predicted errors or smoothness indicators are used as adaptation criteria, as presented in the following sections. Bangerth and Rannacher (2003, Sec. 5.2) described nontrivial strategies on how to decide based on these adaptation criteria, from which we present a commonly used selection.

So called *fixed-error-reduction* or *fixed-fraction* strategies select subsets of cells whose criteria accumulate to a predefined fraction of their global sum. This strategy is only applicable when the sum of all criteria actually has meaning, for example local errors which add up to the global one. Furthermore, it may lead to optimal meshes for several problems, but tends to only adapt very few cells whenever singularities are encountered.

Furthermore, strategies known as *fixed-rate* or *fixed-number* pick predefined fractions of cells with the lowest or highest criteria for adaptation. This allows to predict the growth of the number of cells, but may not lead to an optimal mesh since more cells may be adapted than necessary. We will use this strategy in our investigations presented in Ch. 4 to compare different adaptation strategies at the same growth rate of the mesh.

For either strategy, when using actual errors or error indicators as adaptation criteria, we typically select the subset of cells corresponding to the higher error for refinement, while the subset with the lower error is considered for coarsening. Applicable implementations of these strategies involve binary searches to determine the section of cells relevant for adaptation. For parallel computations, according algorithms have been developed by Burstedde, Ghattas, et al. (2008, Sec. 3.1) and Bangerth, Burstedde, et al. (2012, Sec. 5.1).

With these strategies at hand, we still need quantifiable adaptation criteria. Typically, we are content with error estimates for this purpose since the actual error is not at our disposal. Algorithms to estimate the error from the current finite element approximation will be presented in the following.

3.2.2 Error estimation

The determination of the error for our finite element approximation requires the actual solution to be at our disposal. However, this is not the case in general, and we need to find an alternative measure.

Kelly et al. (1983) worked out an *a posteriori* error estimator for the generalized Poisson equation $-\nabla\cdot(a\nabla u) = f$, where a is a function usually describing material characteristics. They determined an upper bound η_K for the error on each cell by balancing the gradient of the finite element approximation u_{hp} on all faces F of the cell's boundary:

$$\|e_{\mathrm{hp}}\|^2_{H_1(\Omega)} \leq C \sum_{K\in\Omega} \eta_K^2 \quad \text{with} \quad \eta_K^2 = \sum_{F\in\partial K} c_F \int_F \left[a \, \frac{\partial u_{\mathrm{hp}}}{\partial n} \right]^2 \mathrm{d}o\,, \tag{3.4}$$

where C is independent of the solution but depends on a, and

$$\left[a\,\frac{\partial u_{\mathrm{hp}}}{\partial n}\right] = a\,\frac{\partial u_{\mathrm{hp}}}{\partial n_K}\bigg|_K + a\,\frac{\partial u_{\mathrm{hp}}}{\partial n_J}\bigg|_J \tag{3.5}$$

denotes the jump of the approximation's gradient on the face between two adjacent cells K and J. Hence Ainsworth and Oden (1997) attribute this estimator to the class of gradient recovery estimators.

The constant c_F depends on the characteristics of each individual face of the cell. Kelly et al. (1983) originally used the constant $c_F = \frac{h_K}{24\,a_{\min}\,p_K}$ on each face, on which we determine the minimum $a_{\min}$ of the given function. Here, h_K and p_K denote both cell diameter and polynomial degree of the finite element on cell K, respectively. Davydov et al. (2017) proposed a different constant for hp-adaptive FEM: They recommended $c_F = \frac{h_F}{2\,a_{\min}\,p_F}$ with h_F the face diagonal and $p_F = \max(p_K, p_J)$ the largest polynomial degree of adjacent elements K and J on this particular face.

This estimator has been determined for the Poisson equation, but has proven its applicability on other problems as well, where this is no longer meant to be an estimator, but rather an error indicator [20].

We will use these error estimates with the modification for hp-adaptive FEM as adaptation criteria to decide which cells we will adapt. We are still left to decide which type of adaptation we want to apply, i.e., h-adaptation or p-adaptation. In the following sections, we present strategies to do so.

3.2.3 Error prediction

Babuška and Suri (1990) determined upper error bounds for numerical solutions based on the arrangement of finite elements. Both mesh resolution and polynomial degrees of the basis functions have a different, yet quantifiable influence on the error leading to Eq. (3.1).

Their findings are valid not only for the numerical solution on a global scope, but on subsets of the domain as well. Local changes by h- and p-adaptation will thus result in different local error bounds. This motivates a strategy to locally decide which type of adaptation to impose based on the refinement history which has been proposed by Melenk and Wohlmuth (2001): We can predict how the current error will change whenever certain areas of our domain are considered for adaptation in the following iteration. These predicted error estimates allow us to decide whether the choice of adaptation in the previous step was justified, and provide the foundation for it in the next step.

We determine how the error bounds on two different arrangements of finite elements will change by calculating their ratio. For this, we assume that both the actual error and its upper bound change with the same rate, which allows us to equate both ratios. We further assume that the solution is sufficiently regular ($m \gg p \;\Rightarrow\; \mu = p$). The ratio of errors then reads:

$$\frac{\|e_{h_\mathrm{f} p_\mathrm{f}}\|_{H^1(\Omega)}}{\|e_{h_\mathrm{a} p_\mathrm{a}}\|_{H^1(\Omega)}} = \frac{h_\mathrm{f}^{p_\mathrm{f}}}{h_\mathrm{a}^{p_\mathrm{a}}} \left(\frac{p_\mathrm{f}}{p_\mathrm{a}}\right)^{-(m-1)} , \tag{3.6}$$

where subscripts 'a' and 'f' denote the finite element that is currently active or will be active after adaptation, respectively.

If we only consider h-adaptation and leave the polynomial degree of the basis functions unchanged ($p_\mathrm{f} = p_\mathrm{a} \equiv p$), we end up with the classical error bound (Babuška and Suri 1990):

$$\frac{\|e_{h_\mathrm{f} p}\|_{H^1(\Omega)}}{\|e_{h_\mathrm{a} p}\|_{H^1(\Omega)}} = \left(\frac{h_\mathrm{f}}{h_\mathrm{a}}\right)^{p} . \tag{3.7}$$

However, if only p-adaptation is considered and we keep the mesh refinement unchanged ($h_\mathrm{f} = h_\mathrm{a} \equiv h$), the ratio of errors still depends on the regularity of the actual solution which is not at our disposal in general. Following the considerations of Melenk and Wohlmuth (2001), we expect p-adaptation to change the error exponentially with the increment of the polynomial degree:

$$\frac{\|e_{h p_\mathrm{f}}\|_{H^1(\Omega)}}{\|e_{h p_\mathrm{a}}\|_{H^1(\Omega)}} = h^{p_\mathrm{f} - p_\mathrm{a}} \left(\frac{p_\mathrm{f}}{p_\mathrm{a}}\right)^{-(m-1)} \simeq c^{p_\mathrm{f} - p_\mathrm{a}} , \tag{3.8}$$

where c is a constant independent of the cell diameter h.

We suggest a similar approach for the hp-adaptation case as well. The above ratio assumes that the underlying mesh has not been changed. We thus identify Eq. (3.8) with an unaltered cell diameter ($h \equiv h_\mathrm{a}$) in Eq. (3.6) resulting in:

$$\frac{\|e_{h_\mathrm{f} p_\mathrm{f}}\|_{H^1(\Omega)}}{\|e_{h_\mathrm{a} p_\mathrm{a}}\|_{H^1(\Omega)}} \simeq \left(\frac{h_\mathrm{f}}{h_\mathrm{a}}\right)^{p_\mathrm{f}} c^{p_\mathrm{f} - p_\mathrm{a}} . \tag{3.9}$$

Now, we will use these findings to develop an algorithm to predict errors of our finite element approximation. Melenk and Wohlmuth (2001) worked out such an algorithm for hp-refinement, which we will extend to hp-coarsening as well. First, we will now only consider individual cells on our domain rather than the whole domain itself. Further, in practical applications, the actual error on these may be not at our disposal. Instead, we use suitable error indicators $\|e_\mathrm{hp}\|_{H^1(K)} \simeq \eta_K$, assuming that they change at the same rate as the actual error.

We apply our consideration summarized in Eq. (3.9) on any form of adaptation. However, h-adaptation poses an additional challenge, since we have to distribute errors from parent to children cells in case of refinement, or combine them in reverse for coarsening. Here, we will only consider isotropic h-adaptation of quadrilaterals in two and hexahedrals in three dimensions, so that exactly $n_c = 2^{\dim}$ children are assigned to a cell, and the ratio of cell diameters $h_\mathrm{f}/h_\mathrm{a}$ is fixed to be 0.5 for refinement and 2 for coarsening. With respect to the H^1-norm of Eq. (3.2), the square of the predicted error of a refined cell is distributed equally on all of its children, while the square sum of errors of coarsened cells is assigned to their parent. We assign future finite elements with their corresponding polynomial degrees on parent and children cells as described in Sec. 3.1. Last, similar to Melenk and Wohlmuth (2001), we introduce control parameters $\gamma_n, \gamma_h \in (0, \infty)$, as well as $\gamma_p \in (0, 1)$ for all three forms of adaptation, i.e., no, h-, and p-adaptation. We end up with a set of equations which covers all possible combinations for hp-adaptation:

$$\text{no adaptation:} \qquad \eta_K^\mathrm{pred} = \eta_K \, \gamma_n \,, \tag{3.10a}$$

$$p\text{-adaptation:} \qquad \eta_K^\mathrm{pred} = \eta_K \, \gamma_p^{p_{\mathrm{f},K} - p_{\mathrm{a},K}} \,, \tag{3.10b}$$

$$hp\text{-refinement:} \qquad \left(\eta_{K_c}^\mathrm{pred}\right)^2 = n_c^{-1} \left(\eta_{K_p} \, \gamma_h \, 0.5^{p_{\mathrm{f},K_c}} \, \gamma_p^{p_{\mathrm{f},K_c} - p_{\mathrm{a},K_p}}\right)^2 \,, \tag{3.10c}$$

$$hp\text{-coarsening:} \qquad \left(\eta_{K_p}^\mathrm{pred}\right)^2 = \sum_c^{n_c} \left(\eta_{K_c} \left(\gamma_h \, 0.5^{p_{\mathrm{f},K_p}}\right)^{-1} \, \gamma_p^{p_{\mathrm{f},K_p} - p_{\mathrm{a},K_c}}\right)^2 \,. \tag{3.10d}$$

To clarify roles during h-refinement and h-coarsening, we marked parent cells K_p and their children K_c with corresponding subscripts, respectively.

We now have an algorithm to predict the error η_K^pred on each cell K for the next adaptation step on the basis of its current error indicator η_K. For this, we need to know how each cell will be adapted in order to choose the appropriate case from Eq. (3.10) on the basis of the adaptation flags set. We are left to find a suitable criterion on how to actually decide which type of adaptation to apply by comparing the current error indicator to its prediction from the preceding adaptation step.

The original idea of Melenk and Wohlmuth (2001) was to compare the actual error of a cell η_K in an adaptation cycle to its prediction η_K^pred from the previous cycle. On all cells flagged for refinement, they consider h-refinement for $\eta_K > \eta_K^\mathrm{pred}$ and p-refinement otherwise. The motivation behind this particular choice is that we keep the grid resolution fine whenever we suspect a singularity, which is usually indicated by a local error larger than its prediction. We will extend this consideration to work for coarsening as well: For this, we need to pick the according strategy that keeps the cell diameter h_K small for $\eta_K > \eta_K^\mathrm{pred}$, and the polynomial degree p_K large otherwise.

An alternative approach would be to use the *fixed-number* adaptation strategy from above: As indicators for each cell, we calculate the difference of predicted and estimated errors $(\eta_K^\mathrm{pred} - \eta_K)$ for each subset of cells flagged for refinement and coarsening, respectively. On cells to be refined, we consider the fraction corresponding to the largest values for p-adaptation, while for cells to

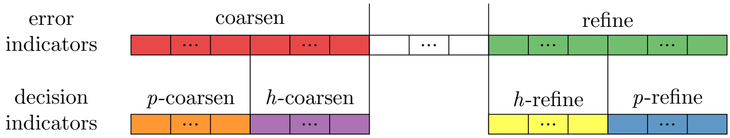


Figure 3.2: *Assignment of cells for hp-adaptation with the fixed-number strategy. Each cell's identifier and its indicator are stored in designated containers, whose entries are sorted in an ascending order of the indicators from left to right. Specified fractions of cells will be marked for adaptation. Large error indicators suggest refinement, while small ones imply coarsening. A large polynomial degree shall be assigned at a large decision indicator, while a fine grid resolution will be applied at a lower one.*

be coarsened, the fraction with the lowest values will be picked. This conforms to the same argumentation as in the original variant. A graphical illustration of the assignment of cells for hp-adaptation using this approach is given in Fig. 3.2, after corresponding indicators for both error and decision have been provided on each cell. We will use this strategy in our applications presented in Ch. 4.

In practice, we need all predicted errors already for the initialization of this method. We provide them with an initial h- or p-adaptation of the mesh, by setting all predicted errors to $\eta_K^{\text{pred}} = 0$ or $\eta_K^{\text{pred}} = \infty$, respectively. We recommend to begin with an initial h-refinement since its error predictor from Eq. (3.7) does not require any information about the solution's regularity and thus yields more reliable results.

This strategy is useful for scenarios to generate a tailored mesh after a few refinement iterations, but lacks applicability for time dependent problems since the refinement history is also connected to the time evolution here. However, this method is well suited to determine the initial grid for initial values of time dependent problems iteratively.

3.2.4 Smoothness estimation

According to Eq. (3.1), we notice that p-adaptation has the largest impact on the error if its corresponding solution is sufficiently regular. Thus as an alternative strategy, determining the smoothness of the finite element approximation presents a reasonable indicator to decide between h- and p-adaptation.

The basic idea to quantify smoothness involves the transformation of a function u into its spectral representation. In one dimension, we expand it into a series of L^2-orthogonal basis functions $(b_k)_{k \in \mathbb{N}_0}$ on an interval $I = [a, b]$ with:

$$\forall x \in I : \qquad u(x) = \sum_{k \geq 0} c_k \, b_k(x) \,, \tag{3.11}$$

$$\forall k, l \in \mathbb{N}_0 : \; k \neq l \,, \qquad \langle b_k, b_l \rangle = 0 \,, \tag{3.12}$$

and identify the smoothness as the rate of decay of the expansion coefficients c_k. In higher dimensional cases, we formulate the expansion in the multi-index notation with tuples $\boldsymbol{k} = (k_1, \ldots, k_{\mathrm{dim}}) \in \mathbb{N}_0^{\mathrm{dim}}$:

$$\forall \boldsymbol{x} \in I^{\mathrm{dim}} : \quad u(\boldsymbol{x}) = \sum_{k_1 \geq 0} \cdots \sum_{k_{\mathrm{dim}} \geq 0} c_{k_1, \ldots, k_{\mathrm{dim}}} b_{k_1, \ldots, k_{\mathrm{dim}}}(\boldsymbol{x}) = \sum_{\boldsymbol{k} \geq 0} c_{\boldsymbol{k}} b_{\boldsymbol{k}}(\boldsymbol{x}) \,, \tag{3.13}$$

where we consider the multi-dimensional expansion as a product of basis functions for every coordinate direction:

$$b_{\boldsymbol{k}}(\boldsymbol{x}) \equiv b_{k_1, \ldots, k_{\mathrm{dim}}}(\boldsymbol{x}) \coloneqq b_{k_1}(x_1) \ldots b_{k_{\mathrm{dim}}}(x_{\mathrm{dim}}) \,. \tag{3.14}$$

In the following, we will present two different ways to estimate the smoothness of the finite element approximation using this method, namely with Legendre and Fourier series expansions.

Mavriplis (1994) was the first to attribute smoothness characteristics to the decay of coefficients from a Legendre series expansion. Legendre polynomials P_k are solutions to the one-dimensional Legendre differential equation on the interval $I = [-1, 1]$:

$$\frac{\mathrm{d}}{\mathrm{d}x} \left((1 - x^2) \, \frac{\mathrm{d}}{\mathrm{d}x} P_k(x) \right) + k \, (k + 1) \, P_k(x) = 0 \,. \tag{3.15}$$

They can be constructed with Rodrigues' formula and fulfill the orthogonality criterion:

$$P_k(x) = \frac{1}{2^k k!} \frac{\mathrm{d}^k}{\mathrm{d}x^k} \left((x^2 - 1)^k \right) \,, \tag{3.16}$$

$$\langle P_k, P_l \rangle = \int_I P_k(x) P_l(x) \, \mathrm{d}x = \frac{2}{2k + 1} \delta_{kl} \,. \tag{3.17}$$

The first Legendre polynomials are depicted in Fig. 3.3. [21]

In the finite element context, all calculations happen on reference cells $\widehat{K} = [0, 1]^{\mathrm{dim}}$. Thus we need to construct a L^2-orthogonal basis on their domain. Furthermore, we require the multi-dimensional variant of Legendre polynomials from Eq. (3.14). The functions are constructed in such a way that they correspond to the orthogonality requirement from Eq. (3.17) [21]:

$$\widetilde{P}_{\boldsymbol{k}}(\boldsymbol{x}) \coloneqq 2^{\mathrm{dim}/2} \, P_{k_1}(2x_1 - 1) \ldots P_{k_{\mathrm{dim}}}(2x_{\mathrm{dim}} - 1) \,, \tag{3.18}$$

$$\left\langle \widetilde{P}_{\boldsymbol{k}}, \widetilde{P}_{\boldsymbol{l}} \right\rangle = \left(\prod_{j \in \boldsymbol{k}} \frac{2}{2j + 1} \right) \delta_{\boldsymbol{kl}} \,. \tag{3.19}$$

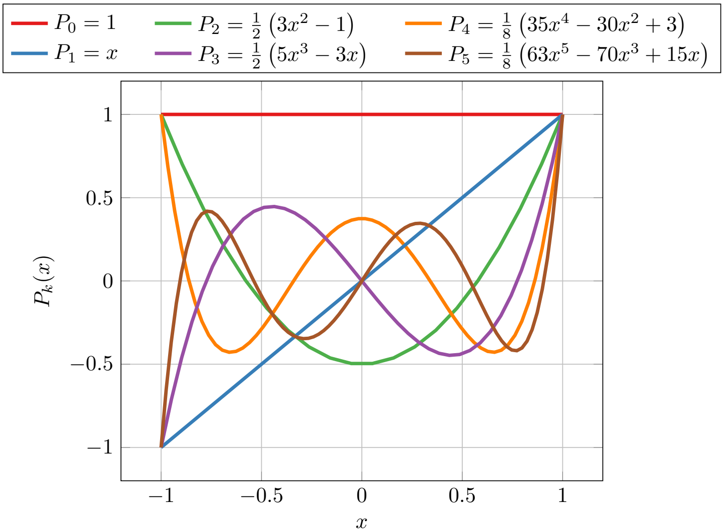


Figure 3.3: *The first six Legendre polynomials.*

Furthermore, the calculation of expansion coefficients requires a mapping of the finite element approximation to the reference cell. This is performed via the transformation $\boldsymbol{x} = \boldsymbol{F}_K(\widehat{\boldsymbol{x}})$, which maps a point $\widehat{\boldsymbol{x}}$ from the reference cell $\widehat{K}$ to the actual cell K. We denote any function φ mapped on the reference cell as $\varphi(\boldsymbol{x}) = \varphi(\boldsymbol{F}_K(\widehat{\boldsymbol{x}})) \equiv \widehat{\varphi}(\widehat{\boldsymbol{x}})$. [22]

With these considerations, we will finally calculate the Legendre expansion coefficients of the finite element approximation u_{hp} on each cell K as follows:

$$c_{\boldsymbol{k}} = \left(\prod_{j\in\boldsymbol{k}} \frac{2j+1}{2}\right) \int\limits_K u_{\mathrm{hp}}(\boldsymbol{x})\, \widetilde{P}_{\boldsymbol{k}}\left(\boldsymbol{F}_K^{-1}(\boldsymbol{x})\right) \mathrm{d}\boldsymbol{x} \tag{3.20}$$

$$= \left(\prod_{j\in\boldsymbol{k}} \frac{2j+1}{2}\right) \int\limits_{\widehat{K}} \widehat{u}_{\mathrm{hp}}(\widehat{\boldsymbol{x}})\, \widetilde{P}_{\boldsymbol{k}}(\widehat{\boldsymbol{x}})\, \left|\det \boldsymbol{J}(\widehat{\boldsymbol{x}})\right| \mathrm{d}\widehat{\boldsymbol{x}}\,, \tag{3.21}$$

with the determinant of the Jacobian $\boldsymbol{J}(\widehat{\boldsymbol{x}}) = \widehat{\nabla}\boldsymbol{F}_K(\widehat{\boldsymbol{x}})$ resulting from the coordinate transformation. [21]

For one-dimensional scenarios, Mavriplis (1994) expanded the finite element approximation in a power series of Legendre polynomials up to the order p_K of the assigned finite element on cell K. Houston and Süli (2005) and Eibner and Melenk (2007) generalized their approach by considering multi-dimensional

Legendre polynomials:

$$\forall \boldsymbol{x} \in K : \quad u_{\mathrm{hp}}(\boldsymbol{x}) \simeq u_{\mathrm{hp},\boldsymbol{k}}(\boldsymbol{x}) = \sum_{0 \leq \|\boldsymbol{k}\|_1 \leq p_K} c_{\boldsymbol{k}} \, \widetilde{P}_{\boldsymbol{k}} \left(\boldsymbol{F}_K^{-1}(\boldsymbol{x}) \right) . \tag{3.22}$$

Eibner and Melenk (2007, Prop. 2) argued that a function is analytic, i.e., representable by a power series, if and only if the absolute values of the expansion coefficients decay exponentially with increasing index $\boldsymbol{k}$:

$$\exists C, \sigma > 0 \quad \forall \boldsymbol{k} \in \mathbb{N}_0^{\mathrm{dim}} : \quad |c_{\boldsymbol{k}}| \leq C \exp\left(-\sigma \|\boldsymbol{k}\|_1\right) . \tag{3.23}$$

Houston and Süli (2005, Sec. 2.4) and Eibner and Melenk (2007, Ch. 4) interpreted the rate of decay σ as a measure for the local smoothness of the finite element approximation and determined it on each cell K by performing a least squares fit on:

$$\forall \boldsymbol{k} \in \mathbb{N}_0^{\mathrm{dim}} : \; 0 \leq \|\boldsymbol{k}\|_1 \leq p_K \, , \quad \ln \left(\max_{\substack{\boldsymbol{l} \\ \|\boldsymbol{l}\|_1 = \|\boldsymbol{k}\|_1}} |c_{\boldsymbol{l},K}| \right) \sim C_K - \sigma_K \|\boldsymbol{k}\|_1 \, , \tag{3.24}$$

where we take the maximum value over all expansion coefficients $c_{\boldsymbol{k}}$ that correspond to the same sum of indices $\|\boldsymbol{k}\|_1$ of the multi-index tuple $\boldsymbol{k}$. Mavriplis (1994) considered a similar approach in one dimension, but only used the last four expansion coefficients for the fit.

Mavriplis (1994) and Eibner and Melenk (2007) treated the rate of decay as a decision criterion for *hp*-refinement by comparing it to a user-provided absolute threshold δ: A decay rate larger than the threshold indicates a good resolution of the finite element basis functions and would entail p-refinement, while a smaller rate would express a bad resolution and thus suggests h-refinement. They considered a threshold of $\delta = 1$ as sufficient.

As an alternative strategy, Bangerth and Kayser-Herold (2009) proposed to use the coefficients of a Fourier series expansion to determine the rate of decay (see also [23]). We will use sinusoidal functions that form an orthogonal basis on the reference cell $\widehat{K} = [0,1]^{\mathrm{dim}}$ as follows:

$$\phi_{\boldsymbol{k}}(\boldsymbol{x}) = \exp\left(-i \, 2\pi \boldsymbol{k} \cdot \boldsymbol{x}\right) \, , \tag{3.25}$$

$$\langle \phi_{\boldsymbol{k}}, \phi_{\boldsymbol{l}} \rangle = \int_{\widehat{K}} \phi_{\boldsymbol{k}}(\boldsymbol{x}) \, \phi_{\boldsymbol{l}}^*(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \delta_{\boldsymbol{k}\boldsymbol{l}} \, . \tag{3.26}$$

We present the real parts of the first few Fourier basis functions in Fig. 3.4.

Suppose a function u on cell K is part of the Hilbert space $H^s(K)$, the following integral must exist:

$$\|\nabla^s u(\boldsymbol{x})\|_{L^2(K)}^2 = \int_K |\nabla^s u(\boldsymbol{x})|^2 \, \mathrm{d}\boldsymbol{x} < \infty \, . \tag{3.27}$$

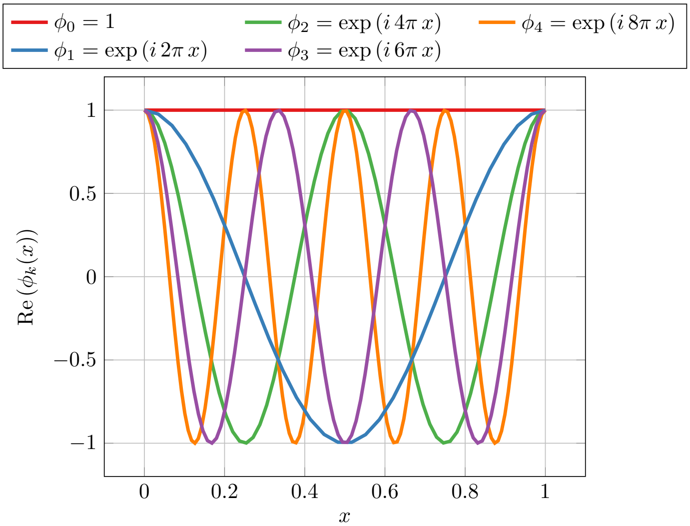


Figure 3.4: *The real parts of the first five basis functions for a one-dimensional Fourier expansion.*

The same condition also applies for its spectral representation $u_{\boldsymbol{k}}$ and can be written as:

$$\|\nabla^s u_{\boldsymbol{k}}(\boldsymbol{x})\|^2_{L^2(K)} = \int_K \left| \sum_{\boldsymbol{k}} (-i\,2\pi\boldsymbol{k})^s\, c_{\boldsymbol{k}}\, \phi_{\boldsymbol{k}}\left(\boldsymbol{F}_K^{-1}(\boldsymbol{x})\right) \right|^2 \mathrm{d}\boldsymbol{x} \tag{3.28}$$

$$= (2\pi)^{2s} \sum_{\boldsymbol{k}} |c_{\boldsymbol{k}}|^2\, \|\boldsymbol{k}\|_2^{2s} < \infty\,. \tag{3.29}$$

The sum is finite only if we require that its summands decay as:

$$\exists \epsilon > 0: \quad |c_{\boldsymbol{k}}|^2\, \|\boldsymbol{k}\|_2^{2s}\, \|\boldsymbol{k}\|_2^{\dim -1} = \mathcal{O}\left(\|\boldsymbol{k}\|_2^{-1-\epsilon}\right)\,. \tag{3.30}$$

The additional factor stems from the fact that, since we sum over all multi-indices $\boldsymbol{k}$ that are located on a dim-dimensional sphere, we actually have, up to a constant, $\|\boldsymbol{k}\|_2^{\dim -1}$ modes located in each increment $\|\boldsymbol{k}\|_2 + \mathrm{d}\|\boldsymbol{k}\|_2$ that need to be taken into account. [23]

With a comparison of exponents, we see that the Fourier coefficients must decay as follows so that the above integral exists:

$$\exists \epsilon > 0: \quad |c_{\boldsymbol{k}}| = \mathcal{O}\left(\|\boldsymbol{k}\|_2^{-\left(s+\frac{\dim}{2}+\epsilon\right)}\right)\,, \tag{3.31}$$

or in other words, when the coefficients decay as $|c_{\boldsymbol{k}}| = \mathcal{O}(\|\boldsymbol{k}\|_2^{-\sigma-\epsilon})$, then the function u is part of $H^{\sigma - \dim/2}(K)$. [23]

We will expand the finite element approximation u_{hp} into a Fourier series on each cell K and will use the local decay rate σ_K of the expansion coefficients as a smoothness indicator. The basis functions of the spectral decomposition are complex-valued for Fourier expansions. Suppose our finite element approximation is real-valued, the expansion coefficients are symmetric $c_{\boldsymbol{k}} = c^*_{-\boldsymbol{k}}$ and we thus only have to calculate all nonnegative multi-indices $\boldsymbol{k}$. The expansion is performed as follows:

$$\forall \boldsymbol{x} \in K : \quad u_{\mathrm{hp}}(\boldsymbol{x}) \simeq u_{\mathrm{hp},\boldsymbol{k}}(\boldsymbol{x}) = \sum_{0 \leq \|\boldsymbol{k}\|_2 \leq p_K} c_{\boldsymbol{k}} \, \phi_{\boldsymbol{k}} \left(\boldsymbol{F}_K^{-1}(\boldsymbol{x}) \right) , \tag{3.32}$$

$$c_{\boldsymbol{k}} = \int\limits_K u_{\mathrm{hp}}(\boldsymbol{x}) \, \phi^*_{\boldsymbol{k}}(\boldsymbol{F}_K^{-1}(\boldsymbol{x})) \, \mathrm{d}\boldsymbol{x} = \int\limits_{\widehat{K}} \widehat{u}_{\mathrm{hp}}(\widehat{\boldsymbol{x}}) \, \phi^*_{\boldsymbol{k}}(\widehat{\boldsymbol{x}}) \, |\det \boldsymbol{J}(\widehat{\boldsymbol{x}})| \, \mathrm{d}\widehat{\boldsymbol{x}} \, . \tag{3.33}$$

We expand our finite element approximation up to a mode that corresponds to the polynomial degree p_K of the currently active finite element. From experience, we decided that this is a suitable choice as results converge from this value on. [24]

With the expansion coefficients at hand, we calculate the decay rate with a least squares fit as follows:

$$\forall \boldsymbol{k} \in \mathbb{N}_0^{\dim} : \ 0 < \|\boldsymbol{k}\|_2 \leq p_K \, , \quad \ln \left(\max_{\substack{\boldsymbol{l} \\ \|\boldsymbol{l}\|_2 = \|\boldsymbol{k}\|_2}} c_{\boldsymbol{l},K} \right) \sim C_K - \sigma_K \ln \left(\|\boldsymbol{k}\|_2 \right) \, . \tag{3.34}$$

We will skip the zeroth mode to avoid the singularity caused by the logarithm.

Bangerth and Kayser-Herold (2009) originally used these smoothness indicators as a decision criterion for *hp*-refinement. They calculate the mean value of the smoothness indicator for all cells flagged for refinement. Whenever the indicator is larger than the average, *p*-refinement is applied in favor of *h*-refinement.

In our investigations, we will use both Legendre and Fourier coefficient decay methods in separate scenarios to estimate the smoothness of our finite element approximation as decision criteria for *hp*-adaptation. Further, we will expand them to utilize *hp*-coarsening as well. For this, we will again use a strategy different from the ones presented by the original authors, namely the *fixed-number* adaptation strategy. On cells to be refined, we consider the fraction corresponding to the largest decay rates for *p*-refinement, while for cells to be coarsened, the fraction with the lowest decay rates will be picked for *p*-coarsening. This corresponds to the same approach of the error prediction strategy and its graphic illustration from Fig. 3.2, and ensures the comparability of all methods.

In practice, we will calculate transformation matrices as auxiliary tools to convert a finite element approximation into its spectral representation. Thus we can perform the transformation by a simple matrix vector product even for higher order elements. For every reference finite element in our collection, a separate transformation matrix has to be generated covering the number of modes corresponding to the polynomial degree p of its basis functions.

For practical reasons, we will only create those matrices on the reference cell $\widehat{K} = [0,1]^{\mathrm{dim}}$. This way we have to perform the costly calculation of transformation matrices only once and will use them to determine the expansion coefficients on every cell K. On the downside, these transformations will only yield applicable results if the cells K are not degenerate, or in other words, when the mapping $\boldsymbol{F}_K$ from the reference cell $\widehat{K}$ to the actual cell K is linear, resulting in a constant Jacobi determinant.

For the Legendre expansion, we determine the coefficients $c_{\boldsymbol{k},K}$ for each cell K via matrix-vector product with the transformation matrix $\widehat{L}_{ij}$:

$$\forall \boldsymbol{k} \in \mathbb{N}_0^{\mathrm{dim}} : \ 0 \leq \|\boldsymbol{k}\|_1 \leq p_K \,, \qquad c_{\boldsymbol{k},K} = \sum_j \widehat{L}_{i(\boldsymbol{k})j}\, u_{j,K} \,, \tag{3.35}$$

$$\widehat{L}_{i(\boldsymbol{k})j} = \left(\prod_{l \in \boldsymbol{k}} \frac{2l+1}{2} \right) \int\limits_{\widehat{K}} \widehat{\varphi}_j(\widehat{\boldsymbol{x}})\, \widetilde{P}_{\boldsymbol{k}}(\widehat{\boldsymbol{x}})\, \mathrm{d}\widehat{\boldsymbol{x}} \,, \tag{3.36}$$

where $u_{j,K}$ denote all entries of the solution vector belonging to the current cell K and $\widehat{\varphi}_j$ are the basis functions of the reference element. The map $i(\boldsymbol{k})$ transforms the multi-index $\boldsymbol{k}$ into an unique integer used as a matrix row index. When using Lagrange finite elements, we will use standard Gaussian quadrature to calculate the integrals with a number of quadrature points of $(p_K + 1)$ in each direction.

The rules to calculate the Fourier expansion coefficients $c_{\boldsymbol{k},K}$ with the corresponding transformation matrix $\widehat{F}_{ij}$ look slightly different:

$$\forall \boldsymbol{k} \in \mathbb{N}_0^{\mathrm{dim}} : \ 0 \leq \|\boldsymbol{k}\|_2 \leq p_K \,, \qquad c_{\boldsymbol{k},K} = \sum_j \widehat{F}_{i(\boldsymbol{k})j}\, u_{j,K} \,, \tag{3.37}$$

$$\widehat{F}_{i(\boldsymbol{k})j} = \int\limits_{\widehat{K}} \widehat{\varphi}_j(\widehat{\boldsymbol{x}})\, \phi_{\boldsymbol{k}}^*(\widehat{\boldsymbol{x}})\, \mathrm{d}\widehat{\boldsymbol{x}} \,. \tag{3.38}$$

Since the Fourier basis functions do not correspond to polynomials, but to sinusoids, we calculate the integrals differently. As a quadrature rule, we will iterate a base quadrature in each direction by a number of times corresponding to the highest mode, which we chose to be p_K. From our experience, a Gaussian quadrature with four points suffices for the role of the base quadrature rule as results converge from this value.

In practice, we noticed that the coefficient decay strategies offer poor results on linear elements. We suspect that linear polynomials alone do not offer sufficient information to make a well-founded statement about the smoothness attribute of the finite element approximation on the cell itself, as we can only work with constants and linear functions in this case. We thus suggest to refrain from using them in this context and work with at least quadratic elements instead.

A variety of different ideas and implementations for smoothness indication strategies have been elaborated in the past. Davydov et al. (2017) also used the method of Legendre coefficients, but determined their decay in each coordinate direction separately and took the minimum over all decay rates as the smoothness indicator. This approach ignores all multi-indices with more than one nonzero entry, which is why we do not consider this approach in our investigations despite its lower computing load.

A different approach is to estimate the Sobolev regularity locally and use it as a means for deciding between h- and p-adaptation. Ainsworth and Senior (1998) presented a way to determine the regularity by solving the problem on smaller patches of the domain, which is a rather expensive approach. On the other hand, Houston, Senior, and Süli (2003) proposed a strategy which estimates the local Sobolev regularity directly from a Legendre series expansion (see also Houston and Süli 2005, Sec. 2.4). We will enhance our collection of decision strategies by this approach in the near future.

3.3 Global data transfer

A major requirement for a general parallel hp-adaptive FEM application is the ability to transfer data from the entirety of all cells or even the complete finite element approximation across adapted meshes. This is especially the case for time dependent or nonlinear problems, in which the solution of a problem on a recently adapted mesh depends on the finite element approximation of the preceding iteration from the previous mesh.

In sequential and parallel applications with shared memory, each process has access to the complete memory set. However in high-performance computing (HPC) applications with distributed memory, we need to transfer data on cells from processes that have previously owned them to processes that will own them after each adaptation or repartitioning.

For communication via Message Passing Interface (MPI), we need to know the size of the data to be transferred, so each process needs to know how much it will send and is going to receive. Since we measure and distribute workload per cell, we know how much data will be sent in case of h-adaptation, since all cells share the same attributes. However, for p-adaptation, the amount of data might vary from cell to cell. A first attempt to make data transfer available on hp-adaptive meshes would be to make the buffer size as large as the biggest data chunk among all cells. However, this would be highly inefficient, since buffers will be way larger than necessary. We need to find a way to send data efficiently.

One possibility would be to use `MPI_Probe` [3] operations to check for sizes of incoming messages before actually receiving them, and decide afterwards on how to receive them. We could make use of this for the transfer of variable size data.

Regardless of this approach, Burstedde (2018, Sec. 5.2) proposed to divide the data transfer into two separate ones that are optimized for each kind of transfer: one for fixed size data transfer, and one for variable size data transfer in this specific order. Information about sizes of the variable data buffers on each individual cell will be sent during the fixed size transfer. In either case, they use nonblocking communication for this purpose, which requires information about the association of cells to their owning process from both the old and the updated mesh as source and destination for the data transfer. Since we exchange data between all cells without exception anyway, we favor this algorithm variant for our implementation.

Many problems to be solved with FEM require multiple data sets to be exchanged between meshes. We will avoid performing the costly transfer multiple times and prepare one consecutive memory buffer with all data to be communicated for both the fixed size and variable size transfer, respectively. This leaves enough flexibility for the transfer mechanism to be independent from the investigated problem.

In the application scope, users will be required to provide functions that prepare data as a consecutive chunk of bytes on each cell, and another one that translates them back. The *pack* function needs to be registered as callbacks and will be triggered during the adaptation process. Data sizes for each cell are available after all *pack* functions have been called. Contiguous memory buffers so created will look like the one depicted in Fig. 3.5. Corresponding *unpack* functions have to be called by the user after adaptation finished and all data structures have been prepared to receive the transferred data. We ensure the correct association of *pack* and *unpack* function calls by assigning integer handles.

For the preparation of data buffers, all refinement indicators need to be terminally set. During both the packing and unpacking process, we determine whether cells will be or have been either refined, coarsened, or left untouched.

Simple data structures assigned on each cell can be easily packed and un-

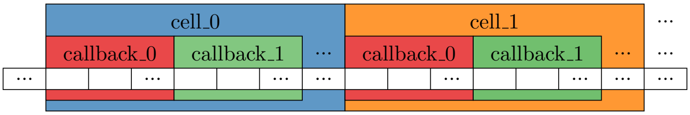


Figure 3.5: *Division of contiguous memory chunks for data transfer. Data is arranged according to the order of the cells. Registering pack functions as callbacks allows to add multiple data chunks from different sources to each cell.*

packed. However in case of h-adaptation, data has to be transferred between parent cells and children. Depending on the context, it is upon the user's decision to provide an appropriate strategy, e.g., divide data of the parent cell equally on all children for refinement and sum data on all children for the parent cell for coarsening.

However, the transfer of complete finite element approximations is slightly more complicated. Here, we need to prepare data on each cell not only depending on the adaptation context, but also on the currently active finite element. In fact, we will already prepare data from the old mesh for the adapted grid in such a way that it just has to be unpacked on the new mesh. Regardless of whether continuous Galerkin (CG) or discontinuous Galerkin (DG) methods are employed, we will always store values of all DoFs on every cell to make sure that all data is available on cells after transfer. Bangerth, Burstedde, et al. (2012) developed an algorithm for transferring the solution across h-adapted meshes in parallel, which will be expanded to work with hp-adaptation in the remaining part of this section.

Once all adaptation indicators have been set, we know how cells will change during the execution of refinement and coarsening, so a corresponding *pack* callback will look as follows: On all active cells which will not be coarsened, we interpolate or project all DoF values of the currently active finite element to the future finite element. On nonactive cells which have active children that will be coarsened, we interpolate or project all DoF values from the currently active finite elements on children to the future finite element of the parent cell, which is determined as the encapsulating finite element space among all children (see Sec. 3.1).

This way, all data has been prepared for the new mesh and has to be distributed on it with the following *unpack* callbacks after hp-adaptation happened and all DoFs have been enumerated on the updated mesh. On every active cell that has not been changed or that is the result of coarsening, we simply extract all DoF values. If an active cell is the result of refinement, we extract all DoF values from its parent cell and interpolate them on the refined cells. All extracted DoF values are left to be copied to the global data container corresponding to the finite element approximation.

In the following, we give a detailed description of the algorithm implemented in the `deal.II` library which is tied to the usage of `p4est` [13] as an oracle and relies on features provided by it. Here, the `deal.II` triangulation is stored independently from the `p4est` mesh.

In the application scope before adaptation is executed, users attach *pack* callback functions to the triangulation and specify whether they qualify for fixed or variable size data transfer.

As soon as the user requests adaptation to be performed, all adaptation indicators will be carried over to the `p4est` master mesh, which will be modified accordingly while maintaining the 2:1 mesh balance. The `deal.II` domain is left untouched.

We store a deep copy of the array of partition markers (Burstedde 2018) from the local `p4est` object, which defines the global partition boundary allowing us to relate each cell to its corresponding subdomain. After repartitioning the `p4est` master mesh, we know each cell's association to its subdomain on both the old and the adapted mesh with the corresponding partition markers, and thus source and destination processes for all MPI communication.

Comparing the meshes of the updated `p4est` object with the `deal.II` triangulation lets us identify how cells have changed. With this information, we are able to prepare data from the old mesh for the new one. We create the contiguous memory buffers for fixed size and variable size data transfer, respectively, by triggering all callback functions that return buffers for the particular data on each cell. In addition, we will store how cells have changed with a corresponding flag and write it to the fixed size buffer.

We determine the sizes of every cell's data pack in each buffer. For fixed size data, we verify the equality of their size on all cells. We store a list with the data size of every cell from the variable size buffer. After that, the fixed size data buffer and the list of sizes from the variable size buffer will be communicated via the optimized fixed size transfer function provided by `p4est` (Burstedde 2018). Last, the variable size data buffer will be transferred using the analogously optimized function after the list of sizes is available.

After adaptation has been performed and all data has been communicated, the user is left to reinitialize all data structures according to the updated mesh and *unpack* the transferred data into them.

With this feature, we are able to send all sorts of cell-related data across adapted meshes, e.g., particle data, quadrature point data, and finite element approximations. Further, we use the above algorithm to transfer active finite element indices internally. To be more precise, future finite element indices will be sent and unpacked as active finite element indices on the adapted mesh.

These contiguous memory chunks can also be used for the purpose of serialization. In case the program shall be interrupted and resumed at a later stage, data will be dumped to the file system. We will create two separate files, each containing either fixed size or variable size data.

Therefore, we need to determine the offset at which each process is supposed to write its local memory buffer into a global contiguous file. Thus, each process needs to know how much memory all preceding processes occupy. For fixed size data, this offset simply translates to the global index of the first cell on this process times the data size per cell. The global index can be determined from the array of partition markers of the `p4est` object (Burstedde 2018). However, in case of variable size data, we need to determine the offset with a prefix sum over the local buffer sizes of all preceding processes using `MPI_Exscan` [3].

To resume the program successfully, we also need to store information about the data size of every cell, which we will prepend to the actual data in each file. In case of fixed size data, an integer corresponding to the data size per cell will be stored. For variable size data, we will collectively write the data size of every local cell in the global file, where each process's offset is determined by the global index of the first cell times the size of an integer.

Finally with this information, we can write all data collectively to the file system using `MPI_File_write_at`, which can later be read from via `MPI_File_read_at` [3]. For the serialization of the global mesh structure, we use the provided save and load functions of `p4est`.

Since all memory is written contiguously and the order of cells on the space-filling curve is independent of the partitioning, we can resume the program with a different number of processes than used during serialization.

Chapter 4

Application to the Laplace problem

With all algorithms and data structures elaborated in Chs. 2 and 3, we now have all features at our disposal to solve partial differential equations with parallel, dynamic hp-adaptive finite element methods (FEM). As a next step, we would like to apply our implementation in the `deal.II` library on a certain exemplary problem that showcases its error performance and parallel scalability. To quantify its capabilities, we want to find a scenario for which an analytic solution u_{sol} is available, allowing us to calculate the actual error of the finite element approximation. This approach for code verification corresponds to the method of manufactured solutions (Salari and Knupp 2000). Our choice for a suitable problem falls on solving the Laplace problem with Dirichlet boundary conditions:

$$-\nabla^2 u(\boldsymbol{x}) = 0 \quad \text{on} \quad \Omega\,, \qquad\qquad u(\boldsymbol{x}) = u_{\mathrm{sol}}(\boldsymbol{x}) \quad \text{on} \quad \partial\Omega\,. \tag{4.1}$$

The choice to study the Laplace problem was not made by chance: We encounter the Laplace equation, or rather the Poisson equation with a non-vanishing right hand side, in many different modeling processes. In the field of electrostatics, the electric potential satisfies the Poisson equation. It is also used to model diffusion processes in time dependent problems. Embedded in time discretization schemes, the Poisson problem has to be solved in each time step for, e.g., heat transfer problems. Further for the simulation of incompressible flows, the coupling of pressure and velocity is governed by the Poisson equation.

Following Eq. (3.1), we expect that p-adaptation is favorable in regions where the solution is regular, while h-adaptation yields better results in regions with discontinuities or singularities. For elliptic problems like the Laplace one, we expect singularities on concave domains (Brenner and Scott 2008, Sec. 5.5).

Mitchell and McClain (2014) presented several benchmarks for *hp*-adaptation that make use of this observation. We decide to showcase our implementation on a two-dimensional domain with a reentrant corner located at the point of origin:

$$\Omega \in \{(r, \theta) \in \mathbb{R}_{\geq 0} \times [0, 2\pi) : 0 \leq \theta \leq \pi/\alpha\} \,, \tag{4.2}$$

with $\alpha \in (1/2, 1)$. With the additional requirement that the solution must be zero along the legs of the reentrant corner, this particular scenario has a general solution which can be formulated in polar coordinates $r = \sqrt{x^2 + y^2} > 0$ and $\theta = \operatorname{arctan2}(y, x)$:

$$\begin{aligned} u_{\mathrm{sol}}(\boldsymbol{x}) &= r^{\alpha} \sin(\alpha\, \theta) \,, && (4.3) \\ \nabla u_{\mathrm{sol}}(\boldsymbol{x}) &= \partial_r u_{\mathrm{sol}}(\boldsymbol{x}) \boldsymbol{e}_r + \frac{1}{r} \partial_\theta u_{\mathrm{sol}}(\boldsymbol{x}) \boldsymbol{e}_\theta && (4.4) \\ &= \alpha r^{\alpha - 1} \left[\sin(\alpha\, \theta) \boldsymbol{e}_r + \cos(\alpha\, \theta) \boldsymbol{e}_\theta \right] \,, && (4.5) \end{aligned}$$

with unit vectors $\boldsymbol{e}_r = \cos(\theta) \boldsymbol{e}_x - \sin(\theta) \boldsymbol{e}_y$ and $\boldsymbol{e}_\theta = \sin(\theta) \boldsymbol{e}_x + \cos(\theta) \boldsymbol{e}_y$. We immediately see that this solution has a singularity near the point of origin for the permitted values of $\alpha \in (1/2, 1)$:

$$\| \nabla u_{\mathrm{sol}}(\boldsymbol{x}) \|_2 = \alpha r^{\alpha - 1} \,, \qquad \forall \alpha \in (1/2, 1) : \quad \lim_{r \to 0} \| \nabla u_{\mathrm{sol}}(\boldsymbol{x}) \|_2 = \infty \,. \tag{4.6}$$

In our testcase, we pick a corner with a right angle with $\alpha = 2/3$ resulting in an L-shaped domain, at which all cells share the same topology to exclude influences from mesh distortion in our benchmark:

$$\Omega_{\mathrm{L}} = [-1, 1]^2 \setminus \{(0, 1) \times (-1, 0)\} \,. \tag{4.7}$$

A depiction of the solution on this particular domain is shown in Fig. 4.1. The three-dimensional variant of this problem is often referred to as the Fichera corner problem.

In this chapter, we will solve the so designed Laplace problem on the L-shaped domain on consecutively refined meshes, and evaluate certain aspects of our implementation, namely the error performance of the decision strategies and their parallel scalability.

Following the usual approach in FEM, we transfer our problem to its weak formulation using continuous Galerkin (CG) methods (Brenner and Scott 2008):

$$(\nabla u, \nabla v) = 0 \,, \qquad \forall v \in V_0 \coloneqq \left\{ v \in H^1(\Omega) : v|_{\partial\Omega} = 0 \right\} \,. \tag{4.8}$$

The shape functions of Lagrange elements will form the basis for the function space $V_h \subset H^1(\Omega)$. Dirichlet boundary conditions are imposed via constraints

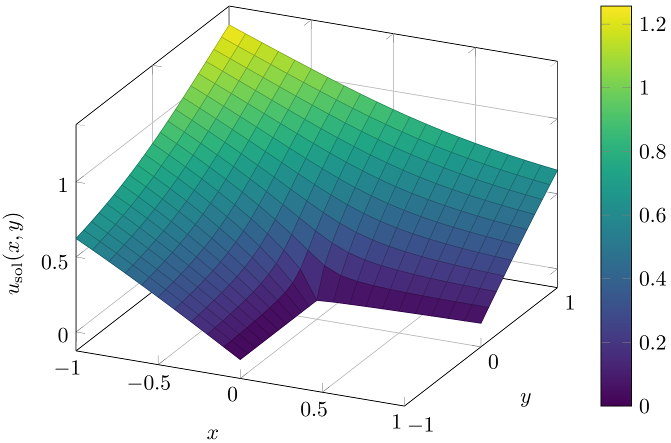


Figure 4.1: *Solution of Eq. (4.3) for the manufactured Laplace problem on the L-shaped domain described by Eq. (4.7).*

on degrees of freedom (DoFs) located on the boundaries. The problem will be solved numerically with an iterative solver based on the conjugate gradient algorithm combined with an algebraic multigrid (AMG) preconditioner.

The `deal.II` library offers interfaces to parallel linear algebra of the third party libraries `PETSc` [12] and `Trilinos` [11] for distributed memory architectures. In our investigations, we choose the latter using their `Epetra` module that handles all data infrastructure, and pick a corresponding solver from their `AztecOO` package as well as their `ML` preconditioner. Compared to an equivalent configuration of `PETSc` modules, the `Trilinos` implementation yields more reproducible results using Message Passing Interface (MPI) ([12], FAQ) and performs faster with higher order polynomials at more advanced refinement iterations according to our experience. For all calculations, we set a tolerance of 10^{-12} relative to the l_2-norm of the right hand side vector of the equation system.

4.1 Error performance

The main motivation to use *hp*-adaptive methods are their superior error convergence characteristics compared to the more common *h*-adaptive methods, provided that the solution is sufficiently regular. We will demonstrate their advantages on our presented scenario and use consecutively adapted meshes to illustrate their error performance in relation to their workload on the numerical example.

The consecutively adapted meshes so created are by far not optimal for the problem, nor is finding such an optimal mesh subject of our investigations. We are rather interested in comparing the performance and results of the decision algorithms presented in Sec. 3.2, and whether they are capable of localizing regions with singularities, i.e., the one at the point of origin for our numerical example. We will compare all types of adaptation at our disposal, i.e., h-, p-, and hp-adaptation. For the latter, we examine all three presented strategies in turn, namely error prediction and smoothness estimation by either Legendre or Fourier coefficient decay.

After solving the linear equation system in each step and calculating the error on the basis of the analytic solution according to Kelly et al. (1983) and Davydov et al. (2017), we use the *fixed-number* strategy to indicate adaptation: The 30 % fraction of all cells with the highest error will be flagged for refinement, and 3 % of those with the lowest error will be marked for coarsening. This allows us to compare the results of each adaptation type under the same conditions, since always the same amount of cells is going to be changed. The idea behind additional coarsening is motivated by the fact that we tend to refine too many cells with error estimators providing an upper bound for the error, or error indicators based on heuristics. We would like to correct this mistake from a previous iteration by coarsening a small amount of cells. The combination of fractions of 30 % for refinement and 3 % for coarsening is a reasonable choice for two-dimensional applications within the `deal.II` library.

Further for hp-adaptive strategies, we need to choose a decision strategy providing corresponding indicators that propose which type of adaptation we want to impose on each cell. Again, we use the *fixed-number* algorithm on all decision indicators as illustrated in Fig. 3.2. This allows us to compare the choices made by each strategy since always the same number of cells is going to be changed in terms of both h- and p-adaptation, respectively. As a first naive approach, we will impose the h-variant on one half of all cells previously marked for adaptation, and the p-variant on the other half. As a second attempt, since we know that we have only one single localized and well-defined singularity in the domain, we are confident to make an educated guess and assign 90 % of flagged cells for p- and the remaining 10 % for h-adaptation. From here on, we refer to the first approach if we call a decision strategy naive, and speak of the second if no such attribute was given.

To set up the numerical example, we start with a mesh consisting of three cells as the coarsest possible representation of our L-shaped domain, which will be globally refined five times to form the initial grid for all investigations in this section. The error prediction strategy forms an exception since it requires a prepended initialization step. In this case, we will begin with four initial refinement steps, solve the equation system, and perform another global refinement so that we have the corresponding predicted errors available. Further for this strategy, control parameters are set to $\gamma_n = 1$, $\gamma_h = 2$, and $\gamma_p = \sqrt{0.4}$, which corresponds to the values used by Melenk and Wohlmuth (2001) and Mitchell and McClain (2014).

We use a collection of Lagrangian finite elements Q_p with polynomial degrees $p \in [2, 7]$ and thus skip linear elements due to our observation that the smoothness estimation algorithms perform poorly with those. All cells will be initially assigned with the lowest order element. In the case of sole h-adaptation all elements will be assigned to Q_2 elements. For pure p-adaptation, we favor p-adaptation over h-adaptation as long as the finite element can be p-adapted, i.e., the current finite element is neither at the top nor the bottom of the hierarchy. We perform a total of twelve consecutive adaptation iterations, so twice as many as there are different finite elements in our collection. In case of the prediction strategy, the total number of iterations including the initialization step is then 13.

All calculations in this section have been carried out on a desktop machine, using an Intel® Core™ i7-4790 processor running at 3.6 GHz with 32 GB of memory. Although this is a quad-core processor offering a total of eight threads utilizing hyper-threading, we will only use a single thread for our calculations, which is sufficient to determine both error and workload. We will deal with parallelization in later sections.

Representatively, we will show the grid and distribution of finite elements after six adaptation cycles of the Legendre coefficient decay strategy with the educated guess approach in Fig. 4.2. All other meshes after equally many adaptation iterations from the other strategies are showcased in App. B. We see that the Legendre strategy is able to locate the singularity in the center by preferring h-adaptive refinement in this section, while using p-adaptation in the other regions. This is also the case for all other strategies, naive or not, as shown in App. B.

We will plot the H^1-error against the workload, which can be measured in two ways: either with the number of DoFs, or the elapsed real time from start to end of our application, which we call wall time. We begin with identifying the workload with the former and show corresponding results in Fig. 4.3.

The double logarithmic representation of our results in Fig. 4.3a reveals analytic convergence for the h-adaptive case. Eq. (3.3) predicts exponential decay for hp-adaptive strategies, which we would like to verify in Fig. 4.3b with a custom scaled plot featuring a logarithmic y-axis and an x-axis scaled with the cube root, which corresponds to the correct exponent in our numerical example. Indeed, we see exponential convergence in the p-adaptive and hp-

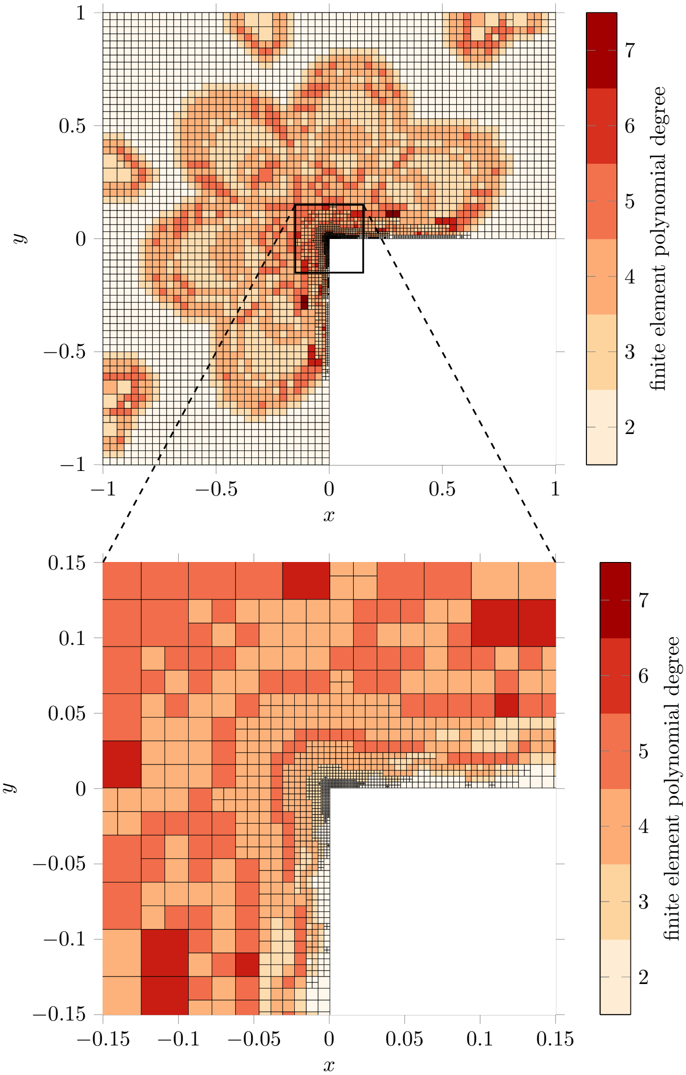
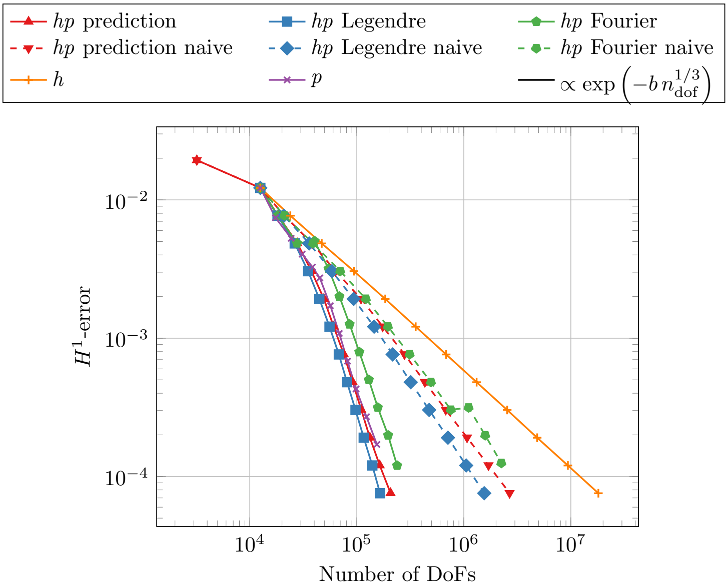
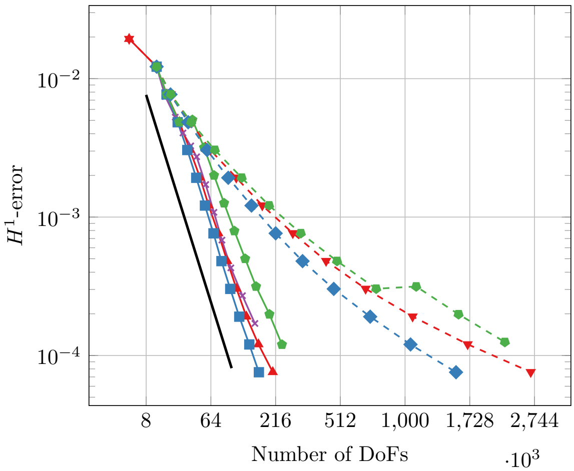


Figure 4.2: *Arrangement of finite elements after six adaptation iterations with hp-adaptation and the smoothness estimation strategy by the decay of Legendre coefficients. The colors represent different polynomial degrees p of the assigned Lagrange elements* Q_p.



(a) *Double logarithmic representation.*



(b) *Custom scaled coordinate system with the y-axis scaled exponentially and the x-axis scaled with the cube root. The thick line corresponds to the reference Eq. (3.3) with* $b = 0.18$*.*

Figure 4.3: *Error performances of several adaptation strategies compared to their workload measured by the number of DoFs.*

adaptive strategies to the proclaimed rate, however the naive approaches miss it. Thus, a high proportion of p-refinement is required to yield exponential decay in this scenario.

The p-strategy shows a similar decay as the hp-adaptive methods. In this strategy, p-refinement will be applied up to the point until it is no longer possible after we reached the highest order element in the hierarchy. With six distinct finite elements in our collection, we will apply h-refinement for the first time after the sixth data point, at which point a cohere drop is observable.

To solve the equation system in the last adaptation cycle, the hp-adaptive methods require a number of DoFs which is lower by a factor of 100 than the h-adaptive methods to achieve the same accuracy and by a factor of ten compared to their naive counterparts. This demonstrates that hp-adaptive methods are the methods of choice for this particular scenario.

In the course of adaptation with the Fourier strategy, some adaptation steps increase the number of DoFs without decreasing the error. We observed that this occurs if the finite element with the highest polynomial degree in our mesh increases from fourth to fifth order. This is perhaps not a causal relation, and we have not yet found the reason for this behavior.

From a practical point of view, the error performance will now be compared to the actual wall time to measure whether we have an economic benefit in using hp-adaptive methods. For each consecutive adaptation cycle, we again plot the calculated error against the workload, which is this time represented by the total run time accumulated over the current and all previous iterations. Each run will be repeated five times and the minimum over the total run time over all runs will be picked to compensate for temporarily high loads on memory bandwidth. The results are shown in Fig. 4.4.

Again after the last adaptation cycle, p- and hp-adaptive methods are about an order of magnitude faster than the h-adaptive variant and thus the most efficient. Interestingly, the wall times of the naive hp-strategies fan out. It appears that a high diversity of finite elements compared with major grid adaptation increases the wall time significantly, and we suspect that our choice of an AMG preconditioner is responsible for this behavior. We will discuss this topic later in the scalability analysis.

The Fourier strategy takes the longest wall time for initialization, during which the Fourier transformation matrices are calculated. This is a costly operation, which requires substantially more quadrature points during calculation than the Legendre equivalent, and is responsible for the observable offset.

Among all hp-strategies of either naive or educated guess category, we do not see major differences in their performance, but overall the Legendre coefficient decay strategy stands out in both categories with the best error per

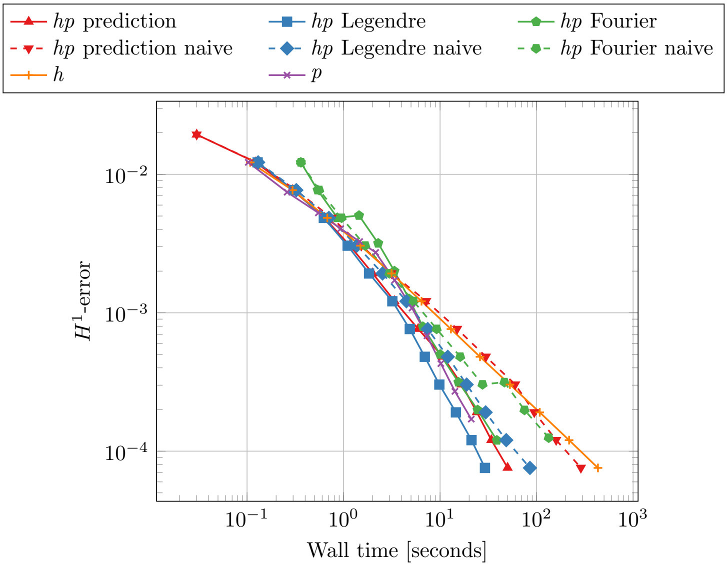


Figure 4.4: *Double logarithmic representation of error performances of several adaptation strategies compared to their workload measured by the wall time.*

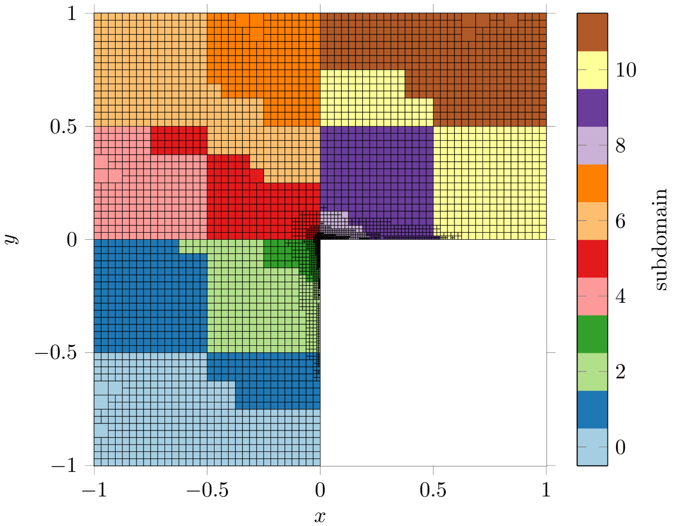
workload performance in either way workload is defined.

Note that the results presented in this section are specific for our numerical example, in which the solution is sufficiently smooth over the entire domain except for the singularity at the origin. The error performance will most probably differ if the presented techniques are applied to a different scenario.

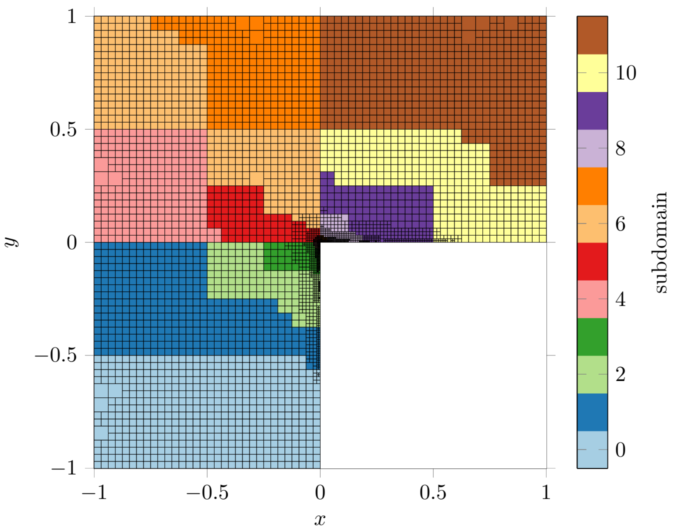
4.2 Load balancing

For parallel computations on distributed memory systems, the global domain is partitioned into several subdomains, each of which is assigned to a single process. Such a mesh decomposition is showcased in Fig. 4.5.

Proper load balancing is necessary for an efficient use of all computational resources. Especially on high-performance computing (HPC) systems with lots of available processors, this is a critical feature. For *hp*-adaptive FEM, we presented approaches for load balancing in Sec. 2.3 by assigning weights to each individual cell and balancing the accumulated weight among all processes. We relate the weight to the number of DoFs on each cell potentiated by an exponent that we will determine in the upcoming investigations. In other



(a) *Constant weighting.*



(b) *Weighting with an individually potentiated number of DoFs, i.e.,* $\propto n_{dofs}^{1.9}$.

Figure 4.5: *Decomposition of the mesh after six iterations with the Legendre coefficient decay strategy on twelve MPI processes with various cell weighting. Each color represents a different subdomain.*

words, cell weights are chosen proportional to n_{dofs}^{c} with the exponent c to be ascertained.

Although all three *hp*-adaptive strategies have demonstrated a similar performance as shown in Sec. 4.1, we pick only one adaptation strategy for our parallel investigations. We choose the smoothness estimation strategy based on the decay of Legendre coefficients as the most efficient one for this purpose.

Investigations are carried out on the JURECA supercomputer (Krause and Thörnig 2018; [14]). Each computing node is equipped with two Intel® Xeon® E5-2680 v3 processors with twelve cores running at 2.5 GHz and either 128 GB, 256 GB or 512 GB of memory. With simultaneous multithreading, a total of 48 threads are available per node. Communication between nodes happens via a Mellanox® EDR InfiniBand high-speed network. More information on the configuration of the supercomputer can be found in [14].

In this section, our investigations are performed on two distinct nodes, which provide a total of 96 threads and involve communication between two physically independent memory segments. We expect that this setup yields representative results that can be extrapolated on even larger problem sizes and different numbers of MPI processes.

Further, we use a 'flat' MPI model: Every thread will be assigned to an individual MPI process and no additional thread parallelization is invoked. Although `deal.II` provides such a feature via Intel® Threading Building Blocks (TBB), we refrain from using it to measure the pure MPI performance for all parallel analyses in this work.

To qualify our problem for parallel computations, we need to increase its size drastically. The problem is initialized with nine global refinements and gets adapted in twelve iterations. For the strategy with the Legendre coefficient decay, this results in a total number of 46,369,440 DoFs, so that each process will be assigned 483,015 DoFs on average. Each type of finite element from the provided collection is represented at least once in the mesh.

This advanced scenario will form the basis of our investigations to see how different weighting exponents affect the wall time, and which one provides a minimum. With serialization, we ensure that each of these runs conforms to the same conditions. Again, to mitigate the impact of temporary slowdowns on the supercomputer due to high loads on memory and network bandwidth, we repeat each run for a total of five times and take the minimum wall time in each category over all runs.

For varying weighting exponents, we compare the wall times of the full adaptation cycle and its relevant sections in Fig. 4.6.

As discussed in Sec. 2.3, the assembly of the equation system and its solution are identified as the critical sections whose wall time is affected by the number

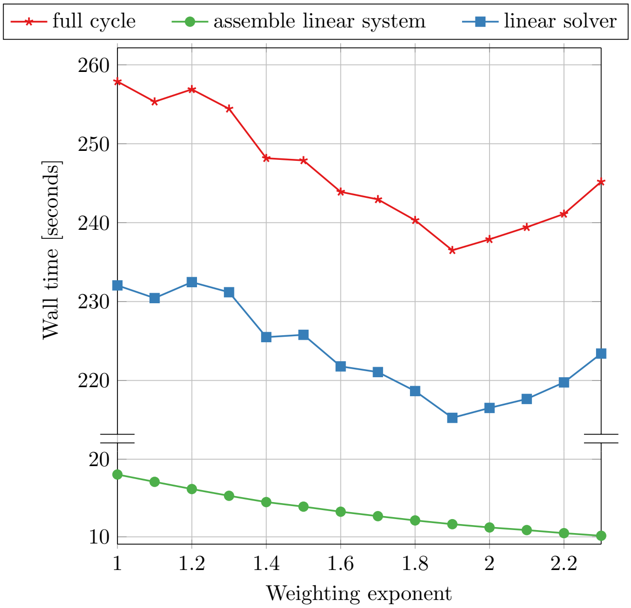


Figure 4.6: *Wall times of a complete adaptation cycle and those parts relevant for load balancing. The problem has about 46 million DoFs and is solved on two nodes or 96 MPI processes. Weights proportional to* n_{dofs}^{c} *will be assigned to each cell with varying exponents* c*.*

of DoFs. We see that the solution of the equation system takes about 90 % of the total wall time and is the crucial factor for proper load balancing. The minimal wall time for both solver and the full cycle is reached with a weighting exponent of $c = 1.9$.

We were dissatisfied to find the minimum wall time of the solver at such a high exponent, since we expected $c = 1$ for an efficient solver. On closer reflection, this is not surprising considering the large number of nonzero entries in the system matrix caused by high order finite elements, which the current implementation of the preconditioner does not handle efficiently. Further, we were expecting a minimum in the assembly at about $c = 2$, but found it was decreasing at even higher exponents. We have no explanation for this behavior.

Anyways, we observed that the total run time does not actually depend particularly strongly on the exponent as it varied by less than 10 % in our investigations. We therefore do not give much importance to the particular choice of cell weights as long as the number of degrees of freedom of each cell is considered.

4.3 High-performance computing scalability

The final part of our investigations relates to the demonstration of the scalability of our algorithms and data structures on HPC systems, for which we will again use the JURECA supercomputer (Krause and Thörnig 2018; [14]).

Again working with successively adapted meshes, we will measure the wall time spent in each particular section of each SOLVE-ESTIMATE-MARK-REFINE iteration, which is supposed to increase linearly with the workload determined by the number of DoFs or decrease linearly with an increasing amount of workers, i.e., number of MPI processes. We distinguish between the following categories in each adaptation cycle similar to Bangerth, Burstedde, et al. (2012):

- *Setup data structures.* Enumerate all DoFs. Determine the sparsity pattern describing locations of nonzero matrix entries. Calculate constraints for hanging nodes and boundary values. Allocate memory for all distributed data structures. Communicate between processes which matrix or vector elements they will write to that they do not own locally.

- *Assemble linear system.* Calculate the individual contribution of each locally owned cell to the global equation system. Exchange data if matrix or vector elements are stored on a different process.

- *Linear solver.* Set up both the AMG preconditioner and the conjugate gradient solver and solve the equation system in parallel.

- *Estimate error.* Calculate the error indicators on locally owned cells on the basis of the current solution. Mark cells for either refinement or coarsening by computing global thresholds.

- *Estimate smoothness.* Calculate the smoothness indicators on the basis of the current solution on locally owned cells marked for either refinement or coarsening. Decide whether h- or p-adaptation is going to be applied by computing global thresholds.

- *Coarsen and refine.* Perform coarsening and refinement and maintain the 2:1 mesh balance on the `p4est` master mesh, followed by its repartitioning. Transfer data between the outdated and updated mesh. Apply all changes made to the master mesh on the `deal.II` grid.

We will pick the parameters and features that have proven to be suitable in our numerical example. Thus, we again choose smoothness estimation by the decay of Legendre coefficients as our hp-decision strategy. For load balancing, cell weighting is imposed proportional to the number of DoFs on each cell potentiated by the exponent $c = 1.9$.

To investigate scaling, we will first consider problems with increasing size solved on a fixed number of MPI processes, which we will realize using consecutive adaptations. We choose two different numbers of computation nodes, namely 16 and 64 nodes with 768 and 3,072 MPI processes in total each.

We initialize the problem with ten initial global refinements and adapt the mesh for a total of eleven iterations with the smaller amount of computing nodes, and twelve iterations for the larger one. In the chosen configuration, all available memory is used on the assigned nodes, so no more adaptation cycles are possible without running out of memory. Again to exclude the influence of the current load on the supercomputer, all runs are performed multiple times and the minimum wall time of each category is taken. This time we repeat each run seven times. The results of scaling on consecutively adapted meshes are shown in Fig. 4.7 up to problem sizes of 2,073,075,769 DoFs.

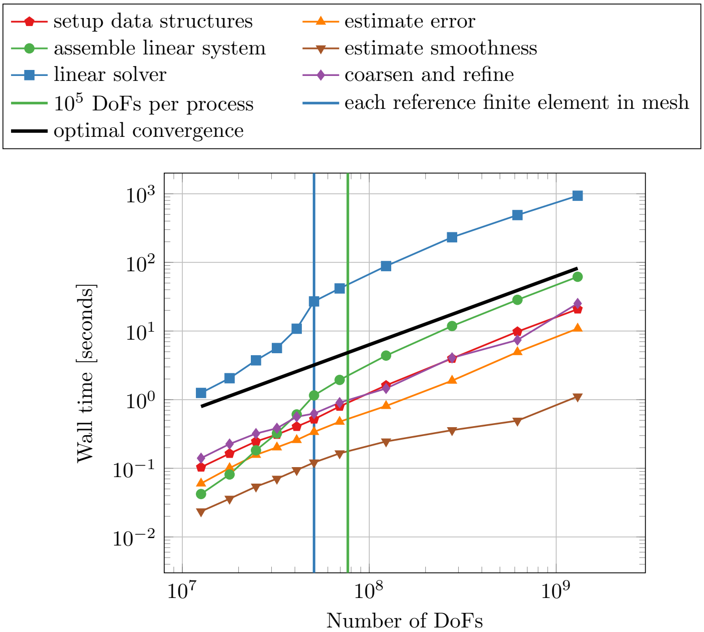
Bangerth, Burstedde, et al. (2012) proclaimed that linear scaling is observable in all categories if the number of DoFs per MPI process exceeds 10^5. We can confirm this observation in our numerical example with parallel *hp*-adaptation as well.

During the first few adaptation cycles in our application, the wall time attributed to the solution category shows a major increase which is more than just linear. After six adaptation cycles, i.e., right of the indicated vertical line in Fig. 4.7, each reference finite element from the collection will be assigned to at least one cell due to the way we configured the scenario, and the aforementioned curve flattens and increases only linearly as expected.

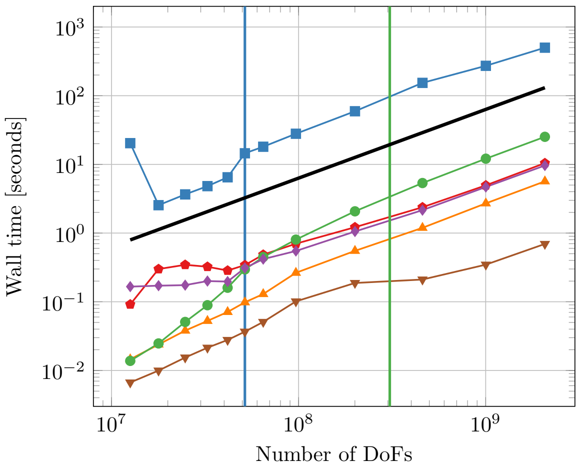
We observe a similar behavior in the number of iterations that the linear solver requires in Fig. 4.8. Here, the number of iterations first increases exponentially and stagnates after said number of adaptation cycles.

We suspect that the rather heterogeneous association of the finite elements by the decision algorithms has a similar effect on the distribution of nonzero entries in the system matrix, for which AMG preconditioners are not designed for. It appears that we could make use of a more suitable preconditioner. Although it was the best option at our disposal at the time of this dissertation, we may think about an alternative to this for future applications.

Mitchell (2010) presented *hp*-multigrid methods for sequential applications. They combined multilevel methods on a geometric hierarchy with those on a hierarchy of finite elements with different polynomial degrees p. The *hp*-adaptive domain will be first relaxed by successively decreasing the polynomial degree of all associated finite elements until only linear ones remain. Next, the geometric multigrid (GMG) method is applied, while results are interpolated back to fi-



(a) *Scaling on 16 nodes or 768 MPI processes.*



(b) *Scaling on 64 nodes or 3,072 MPI processes.*

Figure 4.7: *Scaling for consecutively refined meshes on different numbers of MPI processes. Each MPI process has more than 10^5 DoFs assigned only to the right side of the indicated vertical line. Each finite element is represented at least once in the mesh only to the right side of the designated vertical line.*

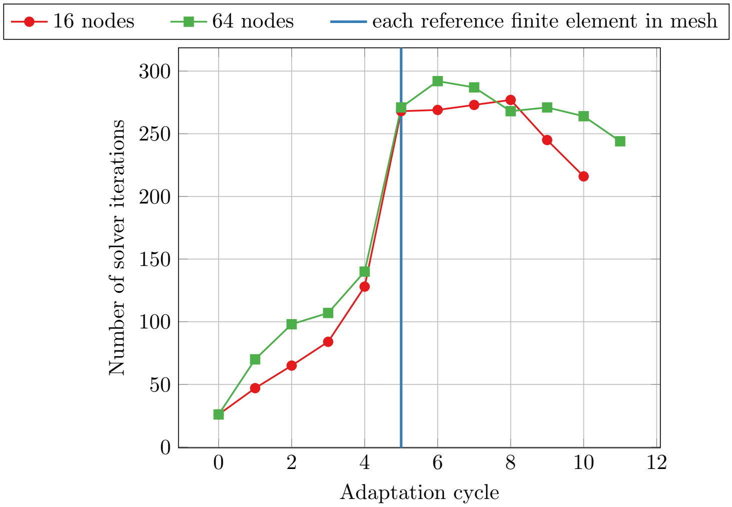


Figure 4.8: *Number of solver iterations at different cycles of consecutive adaptations.*

nite elements of the original polynomial degrees as a last step. The embedding of p-relaxation and p-interpolation in either AMG or GMG preconditioning is promising. Different combinations of multilevel methods in a polynomial, geometric, and algebraic sense are possible, as Fehn et al. (2019) demonstrated on static nonadaptive meshes. Therefore, an even more favorable error to wall time performance of hp- compared to h-adaptive methods are conceivable than presented in Fig. 4.4 and described in Sec. 4.1.

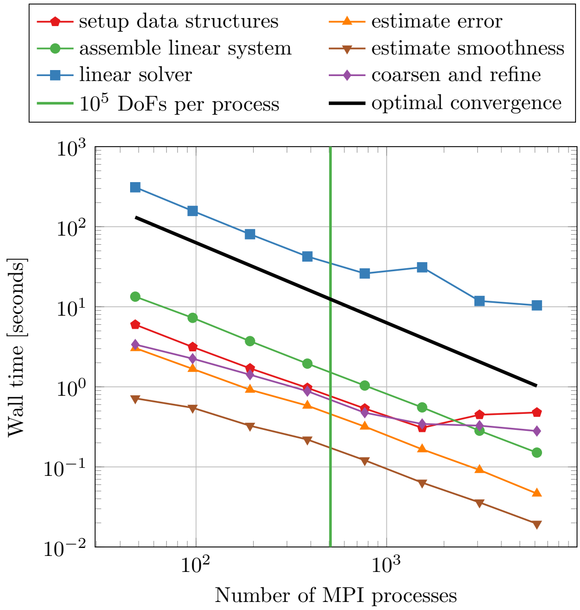
Furthermore, we find a similar behavior of the wall time in the assembly category. The evaluation of finite element shape functions and their derivatives on all quadrature points is an expensive operation, even if it is performed on the reference cell to be projected onto an actual cell by mapping. Furthermore in the context of hp-adaptive methods, this evaluation has to be performed for every single reference finite element in our collection. Specifically for `deal.II`, these values will only be evaluated whenever a cell with the corresponding reference finite element assigned is visited for the first time. Thus, not all of these evaluation objects are calculated until all finite elements are actually represented in the domain. In our application, we unintentionally re-calculated these values again in each adaptation step. Combined with our choice of a decision strategy in favor of p-adaptation, we observe a cohere increase of the wall time until every reference finite element is represented at least once in the domain. For future applications we will move the evaluation of all reference finite element in front of the actual timing investigations.

For strong scaling, problems are set to a fixed size and are solved with an increasing number of MPI processes. This time, we just solve one individual adaptation cycle on a tailored mesh, that has been prepared from a previous run via serialization.

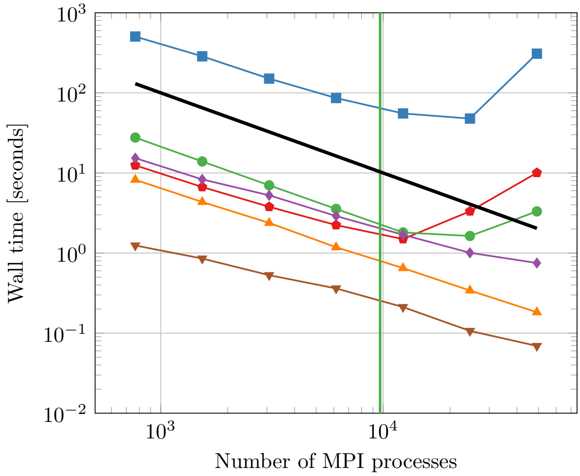
To prepare these meshes, we consider two different scenarios which will be constructed as follows: A smaller scenario is initialized with ten global refinements, and a larger one with twelve refinements. Both will be adapted successively in six adaptation cycles, which results in each reference finite element being represented at least once in the whole domain. This leads to a number of DoFs of 50,736,415 and 969,257,276 in total for the respective scenarios.

With serialization, both problems will be solved at their advanced stage with varying numbers of MPI processes, and the wall times of each section in the program will be recorded. We again repeat each run for a total of seven times and take the minimum wall time in each category, except for the largest run in order to solve the bigger problem on 1,024 nodes or 49,152 MPI processes, which we only repeated five times.

The results of strong scaling are shown in Fig. 4.9. Again, we identify linear scaling whenever the number of DoFs per MPI process exceeds 10^5, which again coincides with the observations of Bangerth, Burstedde, et al. (2012).



(a) *Strong scaling for a fixed problem size of roughly 51 million DoFs.*



(b) *Strong scaling for a fixed problem size of roughly 970 million DoFs.*

Figure 4.9: *Strong scaling for one advanced adaptation cycle at different problem sizes. Each MPI process has more than 10^5 DoFs assigned only to the left side of the indicated vertical line.*

Chapter 5

Summary and outlook

The finite element method (FEM) offers the unique capability of *hp*-adaptive methods with remarkable properties in error convergence relative to workload. However for high-performance computing (HPC), their parallel implementation for large-scale computing architectures with distributed memory via Message Passing Interface (MPI) is difficult.

We presented generic algorithms and data structures for massively parallel *hp*-adaptive FEM, which allow for dynamic changes in both grid resolution and assignment of finite elements. Our findings are independent of the implementation and can be used to enhance any kind of FEM software, provided that their concepts conform to the elementary work of Bangerth and Kayser-Herold (2009) and Bangerth, Burstedde, et al. (2012), which forms the basis of our research.

In this dissertation, we elaborated on the nontrivial parts of combining both parallel *h*-adaptive and sequential *hp*-adaptive methods. The unique enumeration of degrees of freedom (DoFs) and their affiliation with the owning MPI process poses challenges for continuous Galerkin (CG) methods whenever finite elements of similar or different polynomial degree meet on subdomain boundaries. We developed an algorithm for the unique enumeration of DoFs in the parallel *hp*-adaptive context which does not require more costly communication with ghost cells than the *h*-adaptive pendant.

For automatic adaptation, refinement criteria on the basis of error indicators are required to decide which parts of the domain should be adapted. In addition using *hp*-adaptive methods, we also have to select the type of adaptation we would like to impose. We presented several state-of-the-art methods for *hp*-refinement, prepared them for parallel applications, and enhanced them for *hp*-coarsening as well.

Cells are distributed on MPI processes in such a way that the workload is balanced among them, which we ensure by a weighted repartitioning. On each cell, we imposed a simple weight proportional to the number of DoFs potentiated by a factor depending on the investigated problem.

Whenever the mesh itself changes in parallel applications, for example by adaptation, workload needs to be redistributed by repartitioning. Depending on the investigated problem, transferring data from the former to the updated mesh is necessary, for example the finite element approximation itself. Using *hp*-adaptive methods in addition, the amount of data to be transferred might vary by cell. We present a general approach to provide contiguous memory sections which will be exchanged using optimized algorithms presented by Burstedde (2018) for data of fixed and variable size, respectively.

We provided a reference implementation in the `deal.II` library and applied it to the Laplace problem on a L-shaped domain, a common numerical benchmark for *hp*-adaptive methods. We have demonstrated their superior error convergence and shown that our implementation scales on up to 49,152 MPI processes.

Algorithms for parallel *hp*-adaptive FEM capable of handling both CG and discontinuous Galerkin (DG) methods have not yet been prepared in a general framework to this extent before. However, our implementation is still at an early stage of development, and there is still plenty of room for improvement, as described throughout this dissertation. Those aspects that leave room for improvements are outlined in the following.

We observed an unfavorable scaling behavior during the solution of the equation system in our *hp*-adaptive FEM application, which we attribute to our choice of algebraic multigrid (AMG) preconditioning. A preconditioner that also incorporates multilevel methods on a hierarchy of finite elements with different polynomial degrees p will be more efficient and solve the linear equation system in less iterations as investigated by Mitchell (2010). They embedded p-multigrid methods in geometric multigrid (GMG) preconditioning for sequential *hp*-adpative applications. Furthermore, Fehn et al. (2019) combined multilevel methods on hierarchies in a polynomial, geometric, and algebraic sense for parallel FEM on static meshes without adaptation. Future work involves the combination of multilevel methods to make them available for parallel *hp*-adaptive methods as well. This also incorporates parallel h-adaptive GMG preconditioners that have been presented by Clevenger et al. (2019) who also provided an implementation in the `deal.II` library.

We described a handful of decision strategies to choose between types of adaptation. However, there are more strategies worth trying out. Houston, Senior, and Süli (2003) and Houston and Süli (2005) directly determined the regularity of the solution from the coefficients of a Legendre expansion of the finite element approximation. We could use those as decision indicators, or directly set the fitting finite element on the basis of their result.

One could also imagine different decision indicators that are specific to the investigated problem. Hence for computational fluid dynamics, we could relate the decision criteria towards a measure for turbulence for example. The absolute value of the vorticity $\boldsymbol{w} = (\nabla \times \boldsymbol{v})$ as the rotation of the fluid velocity $\boldsymbol{v}$ would make up a good measure. We would prefer h-refinement on turbulent regions indicated by a high vorticity, and p-refinement in laminar regions.

Future work will involve examining the possibility to combine hp-adaptive methods with so called matrix-free methods. Memory access is the current bottleneck on HPC machines. Instead of calculating matrix entries and storing them, it might be faster to calculate them on the fly as they are requested. Combined with single instruction, multiple data (SIMD) instructions or graphics processing unit (GPU) acceleration, this is a highly favorable strategy on current HPC machines. Matrix-free methods have been part of the `deal.II` for a long time (Kronbichler and Kormann 2012), and have been continuously enhanced during the last decade (Kronbichler and Kormann 2019). Furthermore, Munch, Kormann, and Kronbichler (2020) recently published an open-source library named `hyper.deal` using high-order DG methods for high-dimensional partial differential equations, which is built on top of `deal.II` and provides an easy-to-use interface to utilize these methods. The purpose of their framework is to investigate the dynamics of plasmas in nuclear fusion reactors involving shocks, which are modeled using the six-dimensional Vlasov equation. An extension with hp-adaptive methods would be highly promising in any case and would unleash their full potential. A specialized decision strategy for hp-adaptation tied to an observable might be more suitable in the context of plasmas than the general strategies presented in this project.

More generally, this framework can also be used to solve many other problems in continuum mechanics as well, e.g., in structural mechanics and fluid dynamics in general. As a concrete application example in geosciences, convection processes in Earth's mantle can be simulated with the open-source code `ASPECT` (Kronbichler, Heister, and Bangerth 2012; [25]) which builds upon the `deal.II` library. Simulation on a domain of planetary scope yields a lot of workload. Thus `ASPECT` already benefits tremendously from parallel h-adaptive methods, and now also has parallel hp-adaptive methods at its disposal with the results of this dissertation.

We are left to see whether hp-adaptive FEM for distributed memory architectures will be well-received by the community. At the very least, it has been a long requested feature of the `deal.II` library, which was first mentioned in the Google Groups™ `deal.II` mailing list[1] in early 2014 and has been in progress since late 2016 [16].

In the past, Bangerth and Kayser-Herold (2009) provided algorithms and data structures for sequential *hp*-adaptive methods and provided a reference implementation in `deal.II`. They have been widely used for multi-physics problems in `deal.II`, coupling different physical models in different parts of the domain by assigning corresponding finite elements. However, automatic *hp*-adaptation stayed mostly in an experimental state within `deal.II` because of its intricate application. The current interface has been redesigned in this project and simplifies its usage, so that it hopefully becomes a widely used feature in the community.

A good approach to make these features more accessible to all users of the library is to write a dedicated tutorial program as part of the `deal.II` library that showcases the new functionality presented in this dissertation. Tutorial programs are meant to demonstrate certain features of the library and give newcomers a fundamental insight into the numerical and computational background, as well as into implementation details due to extensive documentation. For the demonstration of parallel *hp*-adaptive FEM, we will translate the numerical example from Ch. 4 into a new stand-alone tutorial as a next step.

After all, parallel *hp*-adaptive methods offer promising capabilities, and with all features left to add they are a very challenging yet exciting topic worth to continue working on.

[1] `https://groups.google.com/d/msg/dealii/BmEF75lOA_E/PjyF9F5Uo3UJ`

Appendix A

Enumeration of degrees of freedom: Demonstration

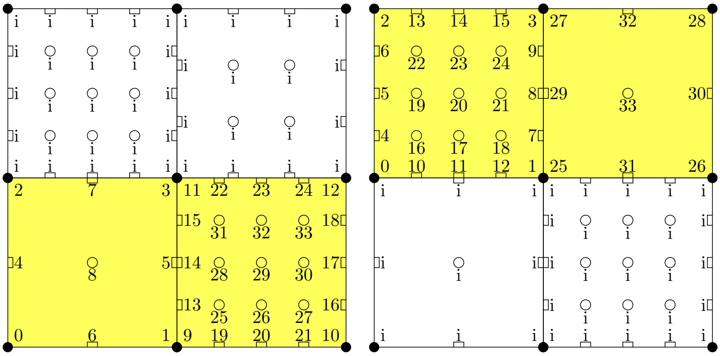
This section delivers a visual demonstration of the enumeration algorithm for degrees of freedom (DoFs) on the corresponding benchmark from Sec. 2.2.

The test case is composed out of four adjacent cells, from which two catty-cornered ones are assigned to the same Lagrangian finite element of either order two or four. The mesh is divided into two subdomains, each containing two neighboring cells of different finite elements. In this configuration, cells are either locally owned or ghost cells. The setup of the benchmark is shown in Fig. 2.4.

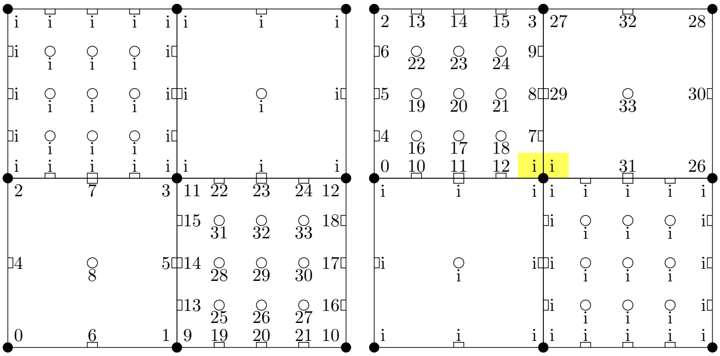
This scenario covers all combinations of adjacent finite elements in the parallel *hp*-adaptive context, which makes it a perfect minimal example. Here, we encounter neighboring cells with similar and different finite elements on the same and another subdomain, with dominating finite elements on either a locally owned or a ghost cell.

For the bulk enumeration of locally owned DoFs, we comply to the scheme that is used in the `deal.II` library. We summarize how DoFs on Lagrange elements are enumerated in two dimensions: We iterate over all cells following a Z-order or Morton space filling curve, starting from the bottom left corner. On each cell, we first enumerate all DoFs on vertices in the same Z-order. Next, all interfaces in the order left, right, bottom, top are enumerated, each starting from the bottom left corner. Finally, all DoFs inside the quadrilateral are enumerated row-wise starting from the bottom left, i.e., lexicographically. [26]

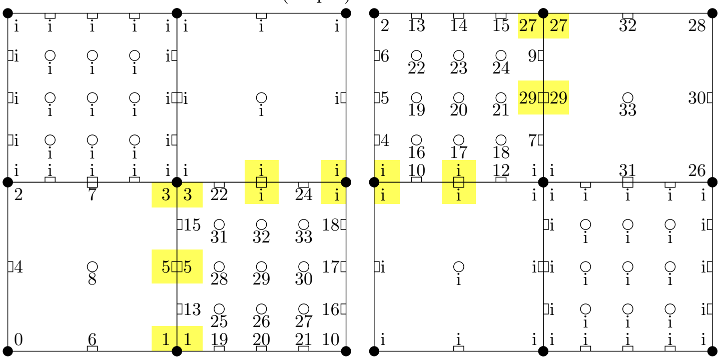
We apply the algorithm step-by-step on this particular example and present its intermediate states in Fig. A.1.



(Step 1) *Local enumeration.*

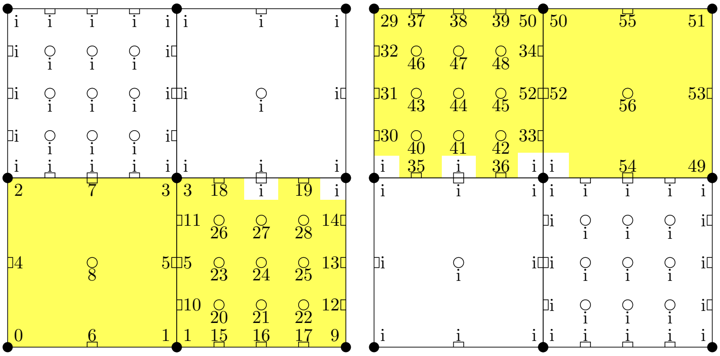


(Step 2) *Tie-break.*

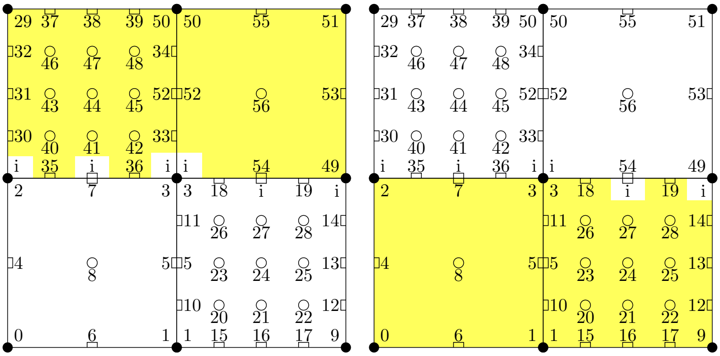


(Step 3) *Unification.*

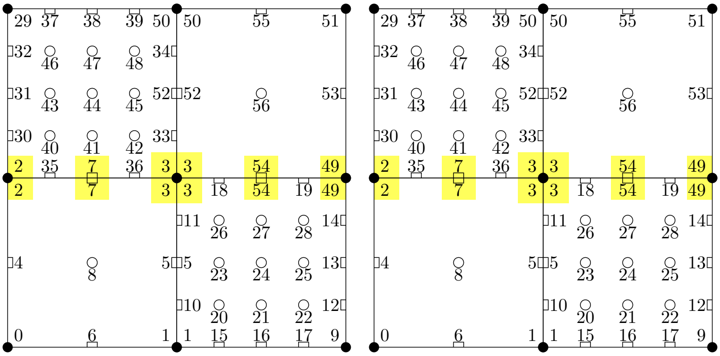
Figure A.1: *Step-by-step demonstration of the enumeration algorithm for DoFs on the benchmark. Changes made at each step are highlighted. The left domain corresponds to the full mesh of the Message Passing Interface (MPI) process with rank 0, the right one belongs to the one with rank 1. The bottom cells are owned by the process with rank 0, and the top cells by the one with rank 1.*



(Step 4) *Global re-enumeration.*



(Step 5) *Ghost exchange.*



(Step 6) *Merge on interfaces.*

Figure A.1: (continued) *Step-by-step demonstration of the enumeration algorithm for DoFs on the benchmark. Changes made at each step are highlighted. The left domain corresponds to the full mesh of the MPI process with rank 0, the right one belongs to the one with rank 1. The bottom cells are owned by the process with rank 0, and the top cells by the one with rank 1.*

Appendix B

Comparison of decision strategies for *hp*-adaptation

In this section, we oppose the adaptation behavior of all utilized strategies applied on our numerical example from Sec. 4.1, i.e., h-, p-, and hp-adaptation with corresponding decision strategies of error prediction and smoothness estimation by the decay of Fourier and Legendre coefficients.

We depict the distribution of finite elements after six consecutive adaptation steps, in which 30 % of cells with the highest error will be refined, and 3 % will be coarsened. For hp-adaptation, half of all those cells will be flagged for h- and p-adaptation as a naive approach. In a second attempt, 90 % of cells marked for adaptation will be flagged for p-adaptation, while the remaining 10 % will be h-adapted. We call the second approach an educated guess for our numerical example.

The meshes of pure h- and p-adaptation are shown in Figs. B.1, B.2. For each hp-adaptation strategy, grids generated with either the naive or educated guess approach are depicted in Figs. B.3, B.4, B.5.

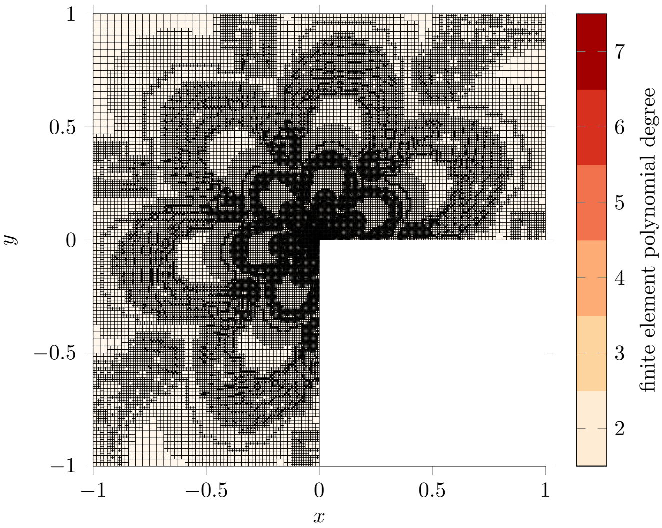


Figure B.1: *Arrangement of finite elements after six adaptation iterations with h-adaptation. The colors represent different polynomial degrees p of the assigned Lagrange elements Q_p.*

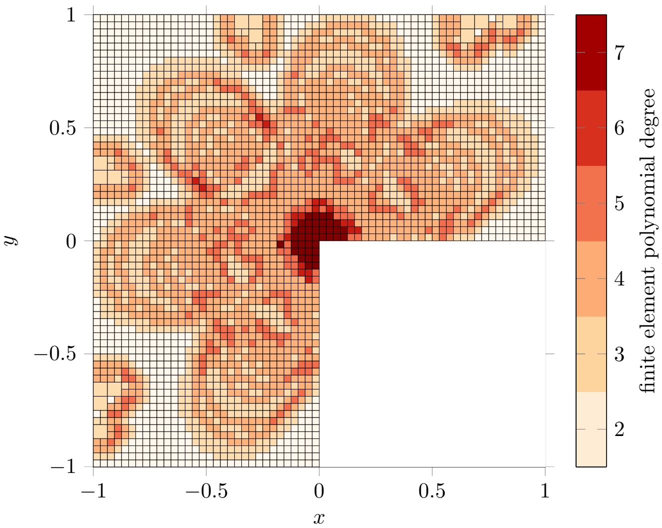
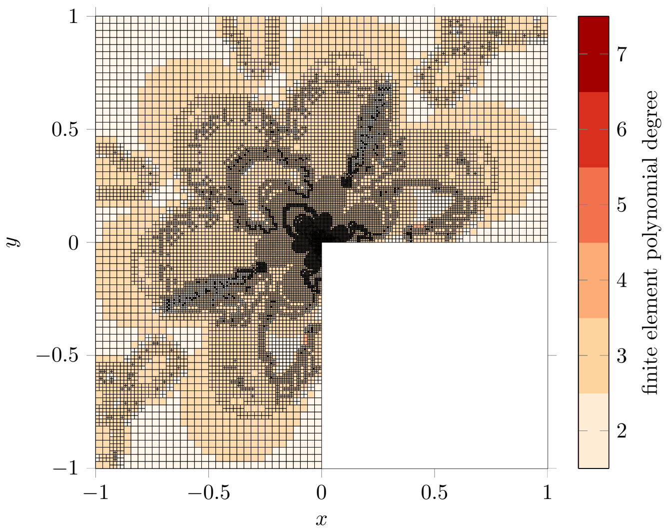
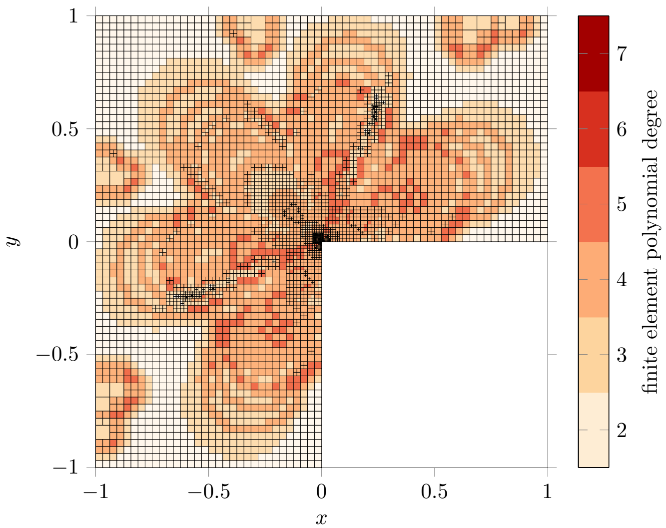


Figure B.2: *Arrangement of finite elements after six adaptation iterations with p-adaptation. The colors represent different polynomial degrees p of the assigned Lagrange elements Q_p.*

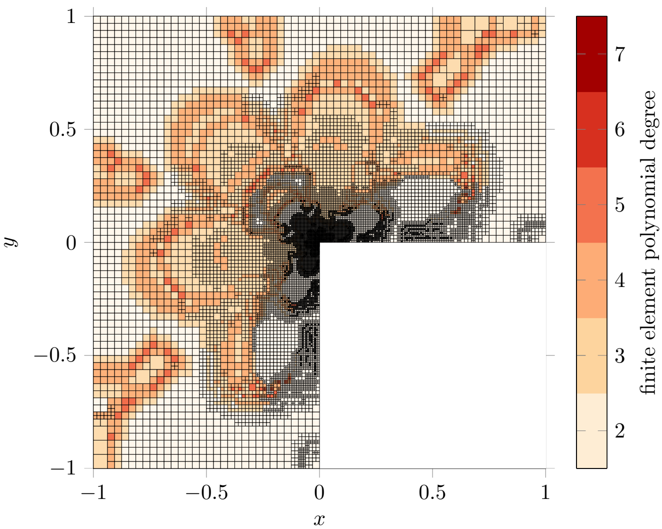


(a) *Naive approach.*

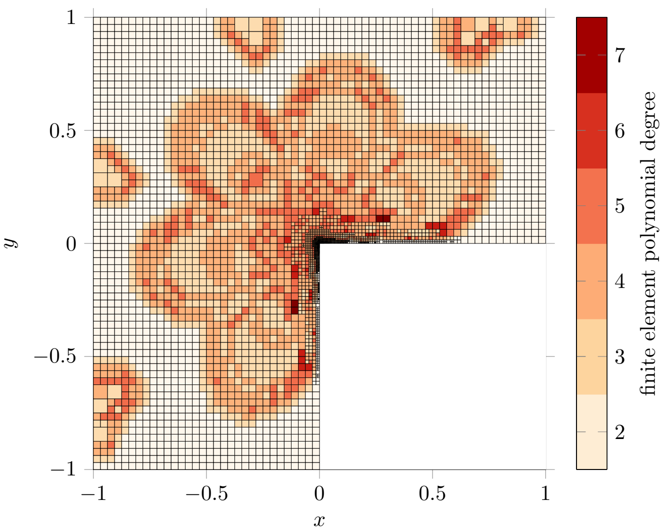


(b) *Educated guess approach.*

Figure B.3: *Arrangement of finite elements after six adaptation iterations with hp-adaptation and the smoothness estimation strategy by the decay of Fourier coefficients. The colors represent different polynomial degrees p of the assigned Lagrange elements Q_p.*

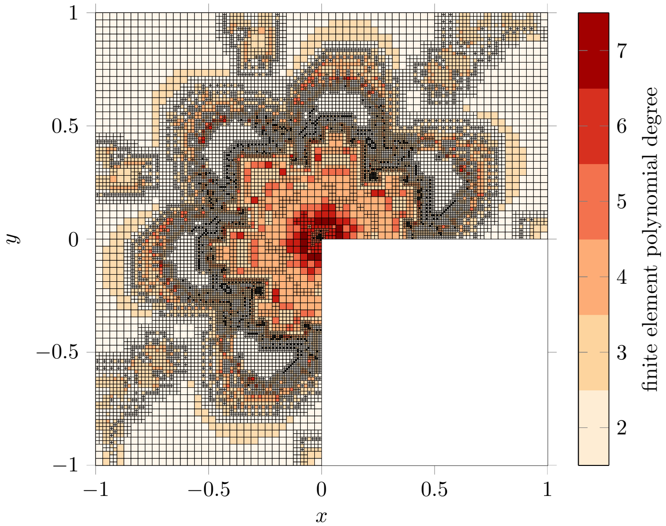


(a) *Naive approach.*

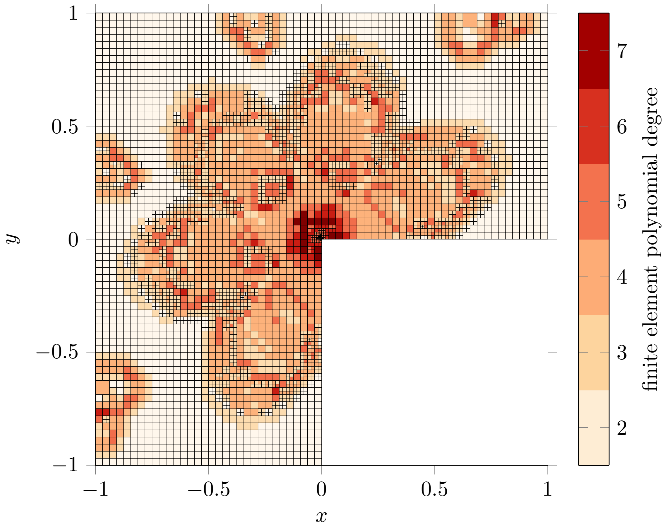


(b) *Educated guess approach.*

Figure B.4: *Arrangement of finite elements after six adaptation iterations with hp-adaptation and the smoothness estimation strategy by the decay of Legendre coefficients. The colors represent different polynomial degrees p of the assigned Lagrange elements Q_p.*



(a) *Naive approach.*



(b) *Educated guess approach.*

Figure B.5: *Arrangement of finite elements after seven adaptation iterations including the initialization step with hp-adaptation and the error prediction strategy. The colors represent different polynomial degrees p of the assigned Lagrange elements Q_p.*

Bibliography

Ainsworth, Mark and J. Tinsley Oden (Mar. 15, 1997). “A posteriori error estimation in finite element analysis”. In: *Computer Methods in Applied Mechanics and Engineering* 142.1-2, pp. 1–88. DOI: 10.1016/S0045-7825(96)01107-3.

Ainsworth, Mark and Bill Senior (Jan. 1998). “An adaptive refinement strategy for *hp*-finite element computations”. In: *Applied Numerical Mathematics* 26.1-2, pp. 165–178. DOI: 10.1016/S0168-9274(97)00083-4.

Arndt, Daniel et al. (2020). “The deal.II library, Version 9.2”. In: *Journal of Numerical Mathematics*. Accepted. URL: https://www.dealii.org/deal92-preprint.pdf (visited on 06/20/2020).

Arnold, Lukas (June 2017). “The ORPHEUS project - Life safety in underground stations”. In: *Book of Abstracts, Poster*. 12th International Symposium on Fire Safety Science (IAFSS). Lund University, Lund Sweden, P130. URL: https://iafss2017.files.wordpress.com/2017/06/book-of-abstract-posters-klar.pdf (visited on 04/03/2020).

Babuška, Ivo and Benqi Guo (July 1996). “Approximation properties of the *h*-*p* version of the finite element method”. In: *Computer Methods in Applied Mechanics and Engineering* 133.3-4, pp. 319–346. DOI: 10.1016/0045-7825(95)00946-9.

Babuška, Ivo and Manil Suri (June 1990). “The p- and h-p versions of the finite element method, an overview”. In: *Computer Methods in Applied Mechanics and Engineering* 80.1-3, pp. 5–26. DOI: 10.1016/0045-7825(90)90011-A.

Balay, Satish et al. (Sept. 2019). *PETSc Users Manual*. ANL-95/11 Rev 3.12. Argonne National Laboratory. URL: https://www.mcs.anl.gov/petsc/petsc-3.12/docs/manual.pdf (visited on 03/29/2020).

Bangerth, Wolfgang, Carsten Burstedde, et al. (Jan. 2012). “Algorithms and Data Structures for Massively Parallel Generic Adaptive Finite Element Codes”. In: *ACM Trans. Math. Softw.* 38.2, pp. 14/1–28. DOI: 10.1145/2049673.2049678.

Bangerth, Wolfgang, Ralf Hartmann, and Guido Kanschat (Aug. 15, 2007). “deal.II - A general-purpose object-oriented finite element library”. In:

ACM Transactions on Mathematical Software (TOMS) 33.4, pp. 24/1–27. DOI: 10.1145/1268776.1268779.

Bangerth, Wolfgang and Oliver Kayser-Herold (Mar. 2009). "Data Structures and Requirements for *hp* Finite Element Software". In: *ACM Trans. Math. Softw.* 36.1, pp. 4/1–31. DOI: 10.1145/1486525.1486529.

Bangerth, Wolfgang and Rolf Rannacher (2003). *Adaptive Finite Element Methods for Differential Equations.* Lectures in Mathematics. ETH Zürich. Birkhäuser Basel. DOI: 10.1007/978-3-0348-7605-6.

Bartlett, Roscoe et al. (Feb. 25, 2017). "xSDK Foundations: Toward an Extreme-scale Scientific Software Development Kit". In: *Supercomputing Frontiers and Innovations* 4.1, pp. 69–82. DOI: 10.14529/jsfi170104.

Brenner, Susanne and Ridgway Scott (2008). *The Mathematical Theory of Finite Element Methods.* 3rd ed. Texts in Applied Mathematics. New York: Springer-Verlag. ISBN: 978-0-387-75933-3. URL: https://www.springer.com/de/book/9780387759333 (visited on 09/18/2019).

Burstedde, Carsten (Mar. 22, 2018). "Parallel tree algorithms for AMR and non-standard data access". In: arXiv: 1803.08432 [cs]. URL: http://arxiv.org/abs/1803.08432 (visited on 08/27/2018).

Burstedde, Carsten, Omar Ghattas, et al. (2008). "Towards adaptive mesh PDE simulations on petascale computers". In: *Proceedings of Teragrid '08.* URL: https://www.researchgate.net/publication/254415344_Towards_Adaptive_Mesh_PDE_Simulations_on_Petascale_Computers (visited on 01/30/2020).

Burstedde, Carsten, Lucas C. Wilcox, and Omar Ghattas (2011). "p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees". In: *SIAM Journal on Scientific Computing* 33.3, pp. 1103–1133. DOI: 10.1137/100791634.

Chalmers, N. et al. (Mar. 2019). "A parallel *hp*-adaptive high order discontinuous Galerkin method for the incompressible Navier-Stokes equations". In: *Journal of Computational Physics: X* 2, pp. 100023/1–22. DOI: 10.1016/j.jcpx.2019.100023.

Ciarlet, Philippe G. (1978). *The Finite Element Method for Elliptic Problems.* Studies in Mathematics and its Applications 4. North-Holland. 529 pp. ISBN: 978-0-08-087525-5. URL: https://www.elsevier.com/books/the-finite-element-method-for-elliptic-problems/ciarlet/978-0-444-85028-7.

Clevenger, Thomas C. et al. (Apr. 5, 2019). "A Flexible, Parallel, Adaptive Geometric Multigrid method for FEM". In: arXiv: 1904.03317 [cs]. URL: http://arxiv.org/abs/1904.03317 (visited on 03/28/2020).

Davydov, Denis et al. (Dec. 12, 2017). "Convergence study of the *h*-adaptive PUM and the *hp*-adaptive FEM applied to eigenvalue problems in quantum mechanics". In: *Advanced Modeling and Simulation in Engineering Sciences* 4, pp. 7/1–23. DOI: `10.1186/s40323-017-0093-0`.

Eibner, T. and Jan M. Melenk (Apr. 2007). "An adaptive strategy for *hp*-FEM based on testing for analyticity". In: *Computational Mechanics* 39.5, pp. 575–595. DOI: `10.1007/s00466-006-0107-0`.

Elman, Howard, David Silvester, and Andy Wathen (June 2014). *Finite Elements and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics*. Oxford University Press. DOI: `10.1093/acprof:oso/9780199678792.001.0001`.

Ern, Alexandre and Jean-Luc Guermond (2004). *Theory and Practice of Finite Elements*. Applied Mathematical Sciences. New York: Springer-Verlag. DOI: `10.1007/978-1-4757-4355-5`.

Fehn, Niklas et al. (Oct. 4, 2019). "Hybrid multigrid methods for high-order discontinuous Galerkin discretizations". In: arXiv: `1910.01900 [physics]`. URL: `http://arxiv.org/abs/1910.01900` (visited on 04/11/2020).

Guo, Benqi and Ivo Babuška (Mar. 1986). "The *h-p* version of the finite element method, Part 1: The basic approximation results". In: *Computational Mechanics* 1.1, pp. 21–41. DOI: `10.1007/BF00298636`.

Heister, Timo (2011). "A Massively Parallel Finite Element Framework with Application to Incompressible Flows". PhD Thesis. Georg-August-Universität Göttingen. URL: `http://hdl.handle.net/11858/00-1735-0000-0006-B6B1-6` (visited on 04/09/2020).

Heroux, Michael A. et al. (Sept. 2005). "An Overview of the Trilinos Project". In: *ACM Trans. Math. Softw.* 31.3, pp. 397–423. DOI: `10.1145/1089014.1089021`.

Houston, Paul, Bill Senior, and Endre Süli (2003). "Sobolev regularity estimation for *hp*-adaptive finite element methods". In: *Numerical Mathematics and Advanced Applications*. ENUMATH 2001. Ed. by Franco Brezzi et al. Milano: Springer Milan, pp. 631–656. DOI: `10.1007/978-88-470-2089-4_58`.

Houston, Paul and Endre Süli (Feb. 4, 2005). "A note on the design of *hp*-adaptive finite element methods for elliptic partial differential equations". In: *Computer Methods in Applied Mechanics and Engineering*. Selected papers from the 11th Conference on The Mathematics of Finite Elements and Applications 194.2-5, pp. 229–243. DOI: `10.1016/j.cma.2004.04.009`.

Jomo, John N. et al. (July 1, 2017). "Parallelization of the multi-level *hp*-adaptive finite cell method". In: *Computers & Mathematics with Applica-*

tions. 5th European Seminar on Computing ESCO 2016 74.1, pp. 126–142. DOI: 10.1016/j.camwa.2017.01.004.

Kaczmarczyk, Łukasz et al. (Jan. 25, 2020a). "MoFEM: An open source, parallel finite element library". In: *Journal of Open Source Software* 5.45, pp. 1441/1–8. DOI: 10.21105/joss.01441.

Kelly, D. W. et al. (1983). "*A posteriori* error analysis and adaptive processes in the finite element method: Part I - error analysis". In: *International Journal for Numerical Methods in Engineering* 19.11, pp. 1593–1619. DOI: 10.1002/nme.1620191103.

Krause, Dorian and Philipp Thörnig (July 25, 2018). "JURECA: Modular supercomputer at Jülich Supercomputing Centre". In: *Journal of large-scale research facilities JLSRF* 4, A132/1–9. DOI: 10.17815/jlsrf-4-121-1.

Kronbichler, Martin, Timo Heister, and Wolfgang Bangerth (Oct. 2012). "High accuracy mantle convection simulation through modern numerical methods". In: *Geophysical Journal International* 191.1, pp. 12–29. DOI: 10.1111/j.1365-246X.2012.05609.x.

Kronbichler, Martin and Katharina Kormann (June 30, 2012). "A generic interface for parallel cell-based finite element operator application". In: *Computers & Fluids* 63, pp. 135–147. DOI: 10.1016/j.compfluid.2012.04.012.

— (Aug. 8, 2019). "Fast matrix-free evaluation of discontinuous Galerkin finite element operators". In: *ACM Transactions on Mathematical Software* 45.3, pp. 29/1–40. DOI: 10.1145/3325864. arXiv: 1711.03590.

Mavriplis, Catherine (1994). "Adaptive mesh strategies for the spectral element method". In: *Computer Methods in Applied Mechanics and Engineering* 116.1-4, pp. 77–86. DOI: 10.1016/S0045-7825(94)80010-3.

Melenk, Jan M. and Barbara I. Wohlmuth (Nov. 2001). "On residual-based a posteriori error estimation in *hp*-FEM". In: *Advances in Computational Mathematics* 15.1-4, pp. 311–331. DOI: 10.1023/A:1014268310921.

Mitchell, William F. (2002). "The Design of a Parallel Adaptive Multi-level Code in Fortran 90". In: *Computational Science - ICCS 2002*. Ed. by Peter M. A. Sloot et al. Vol. 3. Lecture Notes in Computer Science 2331. Berlin, Heidelberg: Springer, pp. 672–680. DOI: 10.1007/3-540-47789-6_70.

— (Apr. 2010). "The *hp*-multigrid method applied to *hp*-adaptive refinement of triangular grids". In: *Numerical Linear Algebra with Applications* 17.2-3, pp. 211–228. DOI: 10.1002/nla.700.

Mitchell, William F. and Marjorie A. McClain (Oct. 2014). "A Comparison of *hp*-Adaptive Strategies for Elliptic Partial Differential Equations". In: *ACM Trans. Math. Softw.* 41.1, pp. 2/1–39. DOI: 10.1145/2629459.

Munch, Peter, Katharina Kormann, and Martin Kronbichler (Feb. 19, 2020). "hyper.deal: An efficient, matrix-free finite-element library for high-dimensional partial differential equations". In: arXiv: 2002.08110 [cs, math]. URL: http://arxiv.org/abs/2002.08110 (visited on 03/12/2020).

Paszyński, Maciej and Leszek Demkowicz (Dec. 2006). "Parallel, fully automatic *hp*-adaptive 3D finite element package". In: *Engineering with Computers* 22.3, pp. 255–276. DOI: 10.1007/s00366-006-0036-8.

Paszyński, Maciej and David Pardo (2011). "Parallel self-adaptive *hp* finite element method with shared data structure". In: *Computer Methods in Material Science* 11.2, pp. 399–405. URL: www.cmms.agh.edu.pl/abstract.php?p_id=361.

Quarteroni, Alfio and Alberto Valli (1994). *Numerical Approximation of Partial Differential Equations*. Springer Series in Computational Mathematics 23. Berlin, Heidelberg: Springer Berlin Heidelberg. DOI: 10.1007/978-3-540-85268-1.

Salari, Kambiz and Patrick Knupp (June 2000). *Code Verification by the Method of Manufactured Solutions*. SAND2000-1444. Sandia National Labs., Albuquerque, NM (US); Sandia National Labs., Livermore, CA (US). DOI: 10.2172/759450.

Zhang, Lin-bo (Mar. 14, 2019). *A Tutorial on PHG*. Academy of Mathematics and Systems Science, Chinese Academy of Sciences. URL: http://lsec.cc.ac.cn/phg/download/tutorial-en.pdf (visited on 03/20/2020).

References

[1] *OpenMP® Application Programming Interface (Version 5.0)* (Nov. 2018). URL: `https://www.openmp.org/` (visited on 02/05/2019).

[2] *Intel® Threading Building Blocks (Version 2019 Update 3)* (Dec. 2018). URL: `https://www.threadingbuildingblocks.org/` (visited on 02/05/2019).

[3] Message Passing Interface Forum (June 2015). *MPI: A Message-Passing Interface Standard (Version 3.1)*. URL: `https://www.mpi-forum.org/` (visited on 02/05/2019).

[4] *The OpenACC® Application Programming Interface (Version 2.7)* (Nov. 2018). URL: `https://www.openacc.org/` (visited on 02/05/2019).

[5] *NVIDIA® CUDA® Toolkit (Version 10.0)* (Oct. 2018). URL: `https://developer.nvidia.com/cuda-zone` (visited on 02/05/2019).

[6] *PHAML: The Parallel Hierarchical Adaptive MultiLevel Project (Version 1.20.0)* (Aug. 28, 2018). URL: `https://math.nist.gov/phaml/phaml.html` (visited on 03/20/2020).

[7] *PHG: Parallel Hierarchical Grid (Version 0.9.4)* (Dec. 30, 2019). URL: `http://lsec.cc.ac.cn/phg/` (visited on 03/20/2020).

[8] Kaczmarczyk, Łukasz et al. (Jan. 24, 2020b). *MoFEM: Mesh Oriented Finite Element Method (Version 0.9.0-joss)*. DOI: `10.5281/zenodo.3627253`.

[9] *deal.II: Differential Equations Analysis Library (Version 9.2.0)* (May 20, 2020). URL: `https://www.dealii.org/` (visited on 06/20/2020).

[10] *xSDK: Extreme-scale Scientific Software Development Kit (Version 0.5.0)* (Nov. 14, 2019). URL: `https://xsdk.info/` (visited on 04/03/2020).

[11] *Trilinos (Version 12.18.1)* (Nov. 11, 2019). URL: `https://trilinos.github.io/` (visited on 02/19/2020).

[12] *PETSc: Portable, Extensible Toolkit for Scientific Computation (Version 3.12.4)* (Feb. 4, 2020). URL: `http://mcs.anl.gov/petsc` (visited on 02/19/2020).

[13] *p4est: Parallel AMR on Forests of Octrees (Version 2.2)* (Feb. 24, 2019). URL: `http://p4est.github.io/` (visited on 03/11/2020).

[14] Forschungszentrum Jülich GmbH. *JURECA: Jülich Research on Exascale Cluster Architectures*. URL: `http://www.fz-juelich.de/ias/jsc/jureca` (visited on 04/09/2020).

[15] Bangerth, Wolfgang. *The deal.II Library: The step-46 tutorial program*. URL: `https://www.dealii.org/9.2.0/doxygen/deal.II/step_46.html` (visited on 07/08/2020).

[16] Bangerth, Wolfgang and Marc Fehling (Nov. 9, 2016). *Make hp::DoFHandler work with parallel::distributed::Triangulation*. GitHub Issue. URL: `https://github.com/dealii/dealii/issues/3511` (visited on 02/19/2020).

[17] Bangerth, Wolfgang. *The deal.II Library: The step-26 tutorial program*. URL: `https://www.dealii.org/9.2.0/doxygen/deal.II/step_26.html` (visited on 07/08/2020).

[18] Kronbichler, Martin and Wolfgang Bangerth. *The deal.II Library: The step-31 tutorial program*. URL: `https://www.dealii.org/9.2.0/doxygen/deal.II/step_31.html` (visited on 07/08/2020).

[19] Fehling, Marc and Denis Davydov (Dec. 8, 2018). *Generalize interface on hp-adaptive methods*. GitHub Issue. URL: `https://github.com/dealii/dealii/issues/7515` (visited on 02/19/2020).

[20] *The deal.II Library: KellyErrorEstimator*. Doxygen Class Template Reference. URL: `https://www.dealii.org/9.2.0/doxygen/deal.II/classKellyErrorEstimator.html` (visited on 07/08/2020).

[21] *The deal.II Library: FESeries::Legendre*. Doxygen Class Template Reference. URL: `https://www.dealii.org/9.2.0/doxygen/deal.II/classFESeries_1_1Legendre.html` (visited on 07/08/2020).

[22] *The deal.II Library: Mapping*. Doxygen Class Template Reference. URL: `https://www.dealii.org/9.2.0/doxygen/deal.II/classMapping.html` (visited on 03/03/2020).

[23] Bangerth, Wolfgang and Denis Davydov. *The deal.II Library: The step-27 tutorial program*. URL: `https://www.dealii.org/9.2.0/doxygen/deal.II/step_27.html` (visited on 07/08/2020).

[24] *The deal.II Library: FESeries::Fourier*. Doxygen Class Template Reference. URL: `https://www.dealii.org/9.2.0/doxygen/deal.II/classFESeries_1_1Fourier.html` (visited on 07/08/2020).

[25] Bangerth, Wolfgang, Juliane Dannberg, et al. (Apr. 29, 2019). *ASPECT: Advanced Solver for Problems in Earth's Convection (Version 2.1.0)*. DOI: `10.5281/zenodo.2653531`.

[26] *The deal.II Library: FE_Q*. Doxygen Class Template Reference. URL: `https://www.dealii.org/9.2.0/doxygen/deal.II/classFE__Q.html` (visited on 07/08/2020).

Band / Volume 31
A study on buoyancy-driven flows: Using particle image velocimetry for validating the Fire Dynamics Simulator
by A. Meunders (2016), xxi, 150 pp
ISBN: 978-3-95806-173-6
URN: urn:nbn:de:0001-2016091517

Band / Volume 32
Methoden für die Bemessung der Leistungsfähigkeit multidirektional genutzter Fußverkehrsanlagen
S. Holl (2016), xii, 170 pp
ISBN: 978-3-95806-191-0
URN: urn:nbn:de:0001-2016120103

Band / Volume 33
JSC Guest Student Programme Proceedings 2016
edited by I. Kabadshow (2017), iii, 191 pp
ISBN: 978-3-95806-225-2
URN: urn:nbn:de:0001-2017032106

Band / Volume 34
Multivariate Methods for Life Safety Analysis in Case of Fire
B. Schröder (2017), x, 222 pp
ISBN: 978-3-95806-254-2
URN: urn:nbn:de:0001-2017081810

Band / Volume 35
Understanding the formation of wait states in one-sided communication
M.-A. Hermanns (2018), xiv, 144 pp
ISBN: 978-3-95806-297-9
URN: urn:nbn:de:0001-2018012504

Band / Volume 36
A multigrid perspective on the parallel full approximation scheme in space and time
D. Moser (2018), vi, 131 pp
ISBN: 978-3-95806-315-0
URN: urn:nbn:de:0001-2018031401

Band / Volume 37
Analysis of I/O Requirements of Scientific Applications
S. El Sayed Mohamed (2018), XV, 199 pp
ISBN: 978-3-95806-344-0
URN: urn:nbn:de:0001-2018071801

Schriften des Forschungszentrums Jülich
IAS Series

Band / Volume 38
Wayfinding and Perception Abilities for Pedestrian Simulations
E. Andresen (2018), 4, x, 162 pp
ISBN: 978-3-95806-375-4
URN: urn:nbn:de:0001-2018121810

Band / Volume 39
Real-Time Simulation and Prognosis of Smoke Propagation in Compartments Using a GPU
A. Küsters (2018), xvii, 162, LIX pp
ISBN: 978-3-95806-379-2
URN: urn:nbn:de:0001-2018121902

Band / Volume 40
Extreme Data Workshop 2018
Forschungszentrum Jülich, 18-19 September 2018
Proceedings
M. Schultz, D. Pleiter, P. Bauer (Eds.) (2019), 64 pp
ISBN: 978-3-95806-392-1
URN: urn:nbn:de:0001-2019032102

Band / Volume 41
A lattice QCD study of nucleon structure with physical quark masses
N. Hasan (2020), xiii, 157 pp
ISBN: 978-3-95806-456-0
URN: urn:nbn:de:0001-2020012307

Band / Volume 42
Mikroskopische Fundamentaldiagramme der Fußgängerdynamik – Empirische Untersuchung von Experimenten eindimensionaler Bewegung sowie quantitative Beschreibung von Stau-Charakteristika
V. Ziemer (2020), XVIII, 155 pp
ISBN: 978-3-95806-470-6
URN: urn:nbn:de:0001-2020051000

Band / Volume 43
Algorithms for massively parallel generic *hp*-adaptive finite element methods
M. Fehling (2020), vii, 78 pp
ISBN: 978-3-95806-486-7
URN: urn:nbn:de:0001-2020071402

Weitere ***Schriften des Verlags im Forschungszentrum Jülich*** unter
http://wwwzb1.fz-juelich.de/verlagextern1/index.asp

IAS Series
Band / Volume 43
ISBN 978-3-95806-486-7

JÜLICH
Forschungszentrum