A Parallel Newton-Krylov-FETI-DP Solver Based on FEAP

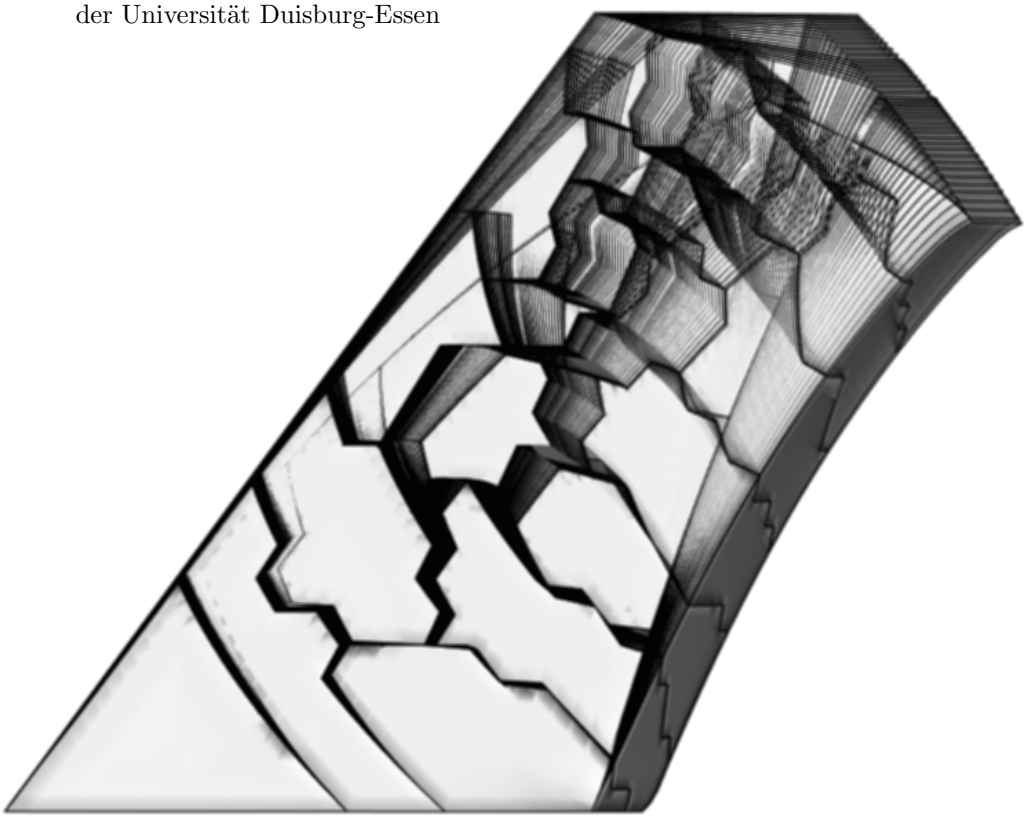
Large-Scale Applications and Scalability for Problems in the Mechanics of Soft Biological Tissues in Arterial Wall Structures

Dipl.-Math. Andreas Fischle
geboren in Bad Mergentheim

Fakultät für Mathematik
Universität Duisburg-Essen

2014

Dissertation im Fach Mathematik
zum Erwerb des Dr. rer. nat.
an der Fakultät für Mathematik
der Universität Duisburg-Essen



Erstgutachter:	Prof. Dr. rer. nat. Axel Klawonn Mathematisches Institut Universität zu Köln
Zweitgutachter:	Prof. Dr. rer. nat. Oliver Rheinbach Institut für Numerische Mathematik und Optimierung Technische Universität Bergakademie Freiberg
Drittgutachter:	Prof. Dr.-Ing. Jörg Schröder Institut für Mechanik, Abteilung Bauwissenschaften Universität Duisburg-Essen

Tag der mündlichen Prüfung: 12.12.2014

Meinen Eltern – Martin und Maria,
meiner Familie – Angelika, Annette, Lars, Régis, Lucas, Samuel und Alisa,
meiner Oma Marie und Tante Berta,
und meinen wundervollen Freunden.

THANK YOU! *Wading somewhere inbetween the established disciplines, each a challenge in its own right; stumbling and staggering into the uncertainty of the New; no results, false results where right are due; just loose ends, year by year, until it all finally comes together; longing for a little star in the darkness – as for most of us, this was no easy task nor time for me. The more I have reason to be grateful! Without the wonderful support, inspiration and distractions that flowed towards me at all times from my family, friends and colleagues, this work would have never seen the light of day ...*

First, I want to thank my advisors for the current thesis project

AXEL KLAWONN *and* OLIVER RHEINBACH

who contributed to its success in different ways but in equal shares. To my principal advisor AXEL KLAWONN *I am very grateful for giving me the outstanding possibility to work on state of the art parallel solvers for applications in continuum mechanics in such a vivid interdisciplinary research environment and for his tireless efforts to provide his students with all that is necessary to thrive scientifically in this domain: the funding, the projects, the research seminar, the access to an entire Cray XT6m supercomputer. I am also grateful for his encouragement and scientific guidance, and also for his support for the current thesis project in mathematics with a strong focus on scientific software engineering; also for many relaxed chats with a good cup of espresso in summer sun at "Die Brücke". My secondary advisor and reviewer* OLIVER RHEINBACH *has been an invaluable inspiration regarding the application of FETI-DP in an HPC setting and for the development of all software components making up the parallel software framework presented in this thesis. He also contributed initial code components – a basis to start from. Without his advice on scalable implementations in general and on his* `FETI-DP-solver` *in particular, I would have been bound to fail. He has excellently motivated, guided and instructed me at all times and I owe him more than words of gratitude. I also want to thank him for many fun distractions! Without* JÖRG SCHRÖDER *and the constant amicable support provided by him and his entire research group, the aim of the present thesis to bridge the software gap between* `FEAP` *and the* `FETI-DP-solver` *could never have been reached. I am very thankful for shared insight into continuum mechanics, for suggesting linear extrapolation as a tuning method, but foremost for the great scientific atmosphere he inspires and for his continuous interest in the progress of my work. I also thank Jörg Schröder for accepting to act as a third reviewer for the current thesis. My mentor and multiple co-author* PATRIZIO NEFF *constantly radiates fascination for the mathematical foundations of continuum mechanics. I thank him for outstanding support and, in particular, for his efforts to help me flourish scientifically as well as personally. And more even, for supporting and believing in me unconditionally – as a friend.* GERHARD STARKE *has been very kind and understanding when integrating me into his research group during the last year of my thesis work and I thank him for his warmhearted support and encouragement as well as for chairing the committee. I also thank the members of the Fakultät für Mathematik for an always amicable ambience, in particular* WOLFGANG RUESS *and* KARL-JOSEF WITSCH.

THANK YOU! (CONT'D) DOMINIK BRANDS *deserves my gratitude for his support at all times, even when pressed for time himself, providing accurate expertise on the innermost intricacies of FEAP and on the finite element implementation of the polyconvex soft-tissue material models which are foundational to the present work. To* ALFIO QUARTERONI *and* SIMONE DEPARIS *at EPFL Lausanne I am thankful for their support during preparation of the D-A-CH research proposal and the great scientific spirit they have brought to CoDELartere! For their support regarding my teaching duties in the Computational Mechanics program I want to thank* ALEXANDER SCHWARZ *and* JOACHIM BLUM. *Further, I want to convey my sincere gratitude to* DAVID STEIGMANN *from UC Berkeley for his benevolent appreciation of my work and for an exceptional offer at just the right time. My colleagues, first at Universität Duisburg-Essen, then at Universität zu Köln have been a great bunch of people to work with – thank you! I feel particularly indebted to* STEFANIE VANIS *for her support during my first years in Essen and in numerical mathematics, for her excellent teaching material and invaluable advice and to* SABRINA GIPPERT *for showing greatness in character proved by abdication – you are a gem! I thank* PATRICK RADTKE *and my office band members* MARTIN LANSER *and* ALEXANDER HEINLEIN *for their support in teaching matters and for common recreational activities. I am also indebted to* ALICE SCHEPERMANN *for support in numerous details. Distinguished thanks go to* DIRK PAULY *for his wonderful way to convey the beauty of mathematics and for very specific but helpful advice; to* WOLFGANG LÖHR *who is just such a great friend – and cook! I am also quite indebted to my assistant* SEBASTIAN HEIN, *to my colleague* BENJAMIN MÜLLER *for taking over lecture duties when it really counted and to* STEFFEN MÜNZENMAIER *for raising my spirits with his cheerfulness. I am very grateful to* NIKLAS NIEMANN *for teaming up to discover the cultural life in Essen – your appearance here made such a difference! I thank* STEPHAN MANDLIER *for his great companionship; also* SLAW OLEINIK *and* MICHAEL SCHOMBURG. *To* JOY ASELMANN *I am very grateful for adamantine motivation and to* LAURA ANDERLE *for being so kind to help in the final phase! I thank* SEBASTIAN BÖLL *for an incredible trip to "The Tip". Merci a mes "colegas" de* GINGER ET CURLY JO*: mon frère et ma sœur d'ésprit* JOSÉ CAMPILLO *et* CHARLOTTE REDLER *à Paris! Ringrazio anche* ILARIA ZAMBONIN, *la regina segreto di Vicenza! Merci à* STANY ET JOSÉPHINE MASSART-KUMPS *pour l'hospitalité belge très apprecié. I thank* ALEXANDER AND ERIOLA POTRYKUS, HANS-JÜRGEN GROTE, MARTIN LAUBINGER, ROBERT VOLLMERT, DIETMAR EBEN, TOBIAS HESSLER, ANITA SELLENT, HEIKE BENECKE, BETTINA SCHOLTE, TILL STEGERS AND VIDYA SWAMINATHAN *and* PETER LIPPERT *for friendship and support. Furthermore, I thank* L'ESCARBILLE, BANDITEN WIE WIR, FACHSCHAFT MATHE, ALJOSCHA MALLMANN, JOCHEN SCHRUMPF *and* CLAUS DOHNALEK. DIANA WIDJAJASAPUTRA *danke ich für Wind in die richtige Richtung. Zu guter Letzt, danke ich auch* <u>*Dir*</u> *für glühende Funken der Inspiration:*

– MEIN KLEINER STERN –

TO ALL MENTIONED HERE AND TO THOSE I MISSED: THANK YOU FOR MAKING THIS POSSIBLE, FOR CHEERING ME UP AND FOR HOLDING ME UPRIGHT WHENEVER IT WAS NECESSARY!

ANDREAS

This page intentionally left blank.

Abstract: An MPI-parallel Newton-Krylov-FETI-DP solver based on `FEAP` is presented together with applications to nonlinear problems in the quasi-static biomechanics of soft biological tissues. The formulation is based on highly nonlinear hyperelastic anisotropic and poly-convex models. High-resolution computations of the wall stresses in patient-specific arterial wall structures subjected to an interior normal pressure in the physiological regime of the blood pressure (up to 500 [mmHg]) are reported together with results on strong scalability. The weak scalability of Newton-Krylov-FETI-DP is investigated for up to 140 million degrees of freedom using 4096 processor cores on a Cray XT6m supercomputer in a series of simple tension tests. An implementation of a new `FEAP`-interface called `libfw` is presented which allows for the flexible unified integration of `FEAP` into other software packages, e.g., into `LifeV`. The modifications done to `FEAP` are dissected and discussed in detail as a case study in order to illustrate possible approaches for the integration of different code components or applications in similar scenarios.

Preface

Software is on the rise in science. The importance of software and modern software development methods in science is certainly on the rise. In recent years, modern software engineering techniques have finally become commonplace in the sciences. All of the large research laboratories employ established techniques and development processes in Software Engineering. Recently, even new ones have been developed, tailored specifically to master and maintain the complexity of scientific software packages. Quite a few scientific journals specifically dedicated to the development of software in the sciences have been founded and their number is increasing. The state of the art in simulation technology requires more skills in software development than ever before – in addition to the scientific training in the domain-specific disciplines. On the other hand, the role of software development in the modern sciences is also subject to active discussion, see e.g., Bangerth and Heister [15].

There is so much excellent software in science: new software and legacy software – contributions of excellent quality based on long-term efforts of exceptional minds – and the merits of coupling established solutions rises with every contribution made. With the limited resources available to scientists and the exceptional ingenuity of the code produced in research there is much attractive potential in coupling the Old and the New. However, this can pose a technical challenge and it can be a worthy contribution to the scientific community. The present work is in part a case study on modularization based on the example of the excellent `F77` legacy software package `FEAP`.[1] We hope to convince the reader of the merits of its reuse as a software component in entirely new contexts.

This thesis has its origin in the interdisciplinary research project

Massiv parallele Simulationen von Arterienwänden: Kontinuumsmechanische Modellbildung und numerische Lösung mittels FETI-Gebietszerlegungsverfahren[a]

(KL 2094/1-{1,2}, SCHR 570/7-{1,2})[b]

[a] Massively parallel simulations of arterial walls: Continuum mechanical modeling and numerical solution via FETI domain decomposition methods (title translated by the author).

[b] The project was supported by the Deutsche Forschungsgemeinschaft (DFG) in two funding periods; common grant proposal by PIs Axel Klawonn and Jörg Schröder.

which inspired a whole series of publications [12, 13, 21–23, 25, 52, 73] and dissertations [20, 24, 55]. Said project marked the beginning of a fruitful interdisciplinary scientific collaboration between multiple research groups continued to this day: the groups led by Axel Klawonn (Universität zu Köln) and Oliver Rheinbach (Technische Universität Bergakademie Freiberg) do work in numerical

[1] Fortran 77 (`F77`) is not a modern programming language and it lacks features essential to modern large-scale software development.

mathematics with a focus on domain-decomposition methods and applications in an HPC-setting (**H**igh **P**erformance **C**omputing) – they use `PETSc`, a mixed `C/C++/F77` software package developed at Argonne National Laboratories [5–7], while the research group led by Jörg Schröder (Universität Duisburg-Essen), including also Daniel Balzani, works on theoretical and computational continuum mechanics – they use a customized version of the `F77` software package `FEAP` developed in Richard L. Taylor's group at the University of California at Berkeley [130].

The rather ambitious common aim of this collaboration are large-scale simulations of realistically modelled multi-physics problems in continuum mechanics. Discretization of the corresponding field equations using appropriate finite element methods easily leads to discretized nonlinear equation systems with millions, hundreds of millions, or even billions of unknowns. The common vision is easily distilled: to solve these equations using highly scalable parallel implementations of domain decomposition-based nonlinear solvers targeted for the current and next generation of massively parallel supercomputers. A long term effort and the progress is made in small steps. Let us shortly summarize the original research focus of the first common project.

Anisotropic models for arterial walls and FETI-DP: As reported by the World Health Organization, cardiovascular diseases are by far the number one cause of death world wide. The World Health Report (2004) lists a death rate of 29.3% for both sexes for the category of cardiovascular diseases which includes arteriosclerosis [106]. Some surgical procedures for atherosclerotic arteries, e.g., balloon angioplasty, are mechanical processes. This motivated much research on the simulation of the mechanics of soft biological tissues in this context. Of particular interest are the stresses in the arterial walls due to mechanical deformation of the wall structure which are correlated to the risk of rupture of plaque components. The latter may cause embolies during surgery. For the simulation, realistic patient-specific arterial wall geometries are a necessary prerequisite.

Expertise on state of the art modelling, parameter fitting and finite element techniques for soft biological tissues based on anisotropic polyconvex hyperelastic formulations were contributed to the present work by the PIs Jörg Schröder and Daniel Balzani. For parallel simulations in HPC, highly scalable and robust domain-decomposition based linear solvers, in particular a `FETI-DP-solver` developed by Oliver Rheinbach in his dissertation [110], were contributed by the PIs Axel Klawonn and Oliver Rheinbach. Based on these, Newton-Krylov-FETI-DP-solvers were implemented for Brands, Klawonn, Rheinbach, and Schröder [23] and developed in multiple directions, e.g., using Augmented Lagrange methods for the incompressibility constraint in Brinkhues, Klawonn, Rheinbach, and Schröder [25]. The application of FETI-methods to problems in finite continuum mechanics was a new scenario which was found to pose significant new challenges. Novel techniques had to be developed. First advancements regarding the finite element modelling and the design of robust parallel scalable preconditioners based on FETI-DP in this context have been contributed to numerical mathematics, scientific computing, computational structural mechanics and to medicine.

For the development of the simulations envisaged in the DFG-funded projects "Massively parallel simulations of arterial walls: Continuum mechanical modeling and numerical solution via FETI domain decomposition methods (KL 2094/1-{1,2}, SCHR 570/7-{1,2})" it was a fundamental prerequisite to couple the software frameworks

`FEAP` [**sequential**] and the **`FETI-DP-solver`** [**parallel**]

in order to meet the formulated research goals – a typical problem in Software Engineering. This posed a significant challenge, because `FEAP` was designed as a standalone application. It was never intended to be used as a software library providing material models such that they can be flexibly

integrated into other programs. It was designed to be the master application reaching out to its extensions. In a first step, both components were coupled in a semi-parallel scenario where a sequential instance of `FEAP` provided the control flow and assembly. The semi-parallel approach used in [23], although an important, necessary and successful intermediate step, was not parallel scalable, finally necessitating a true unified integration of the software components. To this end, previous attempts for a fully parallel solution were undertaken but fell short of expectations with respect to the long term maintainability, the ease of use and robustness of the solution. In the context of the present thesis, a *fully MPI parallel and scalable solution* has finally been achieved, and more ...

Contributions:

1. We contribute a new MPI parallel software framework unifying the highly parallel scalable and robust linear `FETI-DP-solver` presented in the doctoral dissertation of Oliver Rheinbach [110] with a modified distribution of `FEAP` developed by Jörg Schröder and his research group in a novel Newton-Krylov-FETI-DP solver called `mparfeap`. A new `FEAP` interface called `libfw` is presented which allows to use `FEAP` as a flexible software library in other programs. We also present an analysis of the various plans of attack which have been considered by the present author, or previously by Oliver Rheinbach, Dominik Brands and Daniel Balzani, in order to integrate `FEAP` and the `FETI-DP-solver` to allow for massively parallel and highly scalable simulations. Further, we have tried to present the software modifications made to `FEAP` in some detail to spark inspiration in other developers in a similar situation, i.e., trying to couple the Old and the New.

2. New computational results were obtained for massively parallel simulations based on Newton-Krylov-FETI-DP for structured and unstructured decompositions applied to problems in the biomechanics of soft tissues. High-resolution computations of arterial wall stresses are presented together with strong and weak parallel scalability experiments. All computations were run on the Cray XT6m supercomputer at Universität Duisburg-Essen. Results for discretized problems with up to 140 million degrees of freedom in the structured case obtained using a maximum of 4096 CPU cores on the full machine are presented. Alongside, the combined effects of two simple and computationally cheap but quite effective tuning strategies for load stepping are presented.

Furthermore, we note that the software component `libfw` is currently used for **f**luid-**s**tructure **i**nteraction (FSI) in a new joint DFG/SNSF D-A-CH project:

Domain-Decomposition-Based Fluid Structure Interaction Algorithms for Highly Nonlinear and Anisotropic Elastic Arterial Wall Models in 3D[a]

[a]This project is supported by the Deutsche Forschungsgemeinschaft (DFG) and the Swiss National Fond (SNF) within a D-A-CH proposal; common grant proposal by PIs Axel Klawonn, Oliver Rheinbach, Alfio Quarteroni, Simone Deparis, Daniel Balzani, and Jörg Schröder.

In this ongoing project, the `FEAP` wrapper library `libfw` presented in Chapter 2 is used as a bridge component. This module allows to use `FEAP` for the assembly of the quasi-static part in the fully time-dependent ALE-based fluid-structure interaction (FSI) simulations of arterial wall stresses in atherosclerotic arteries in `LifeV` [91][2].

[2]LifeV is an actively developed software framework for the life sciences currently developed at:
École Polytechnique Fédérale de Lausanne (CMCS), Switzerland;
Politecnico di Milano (MOX), Italy;

Note that the finite element discretizations for the St. Venant-Kirchhoff and Neo-Hooke material models available in both `FEAP` and `LifeV` have been verified to be consistent up to numerical tolerance by Alexander Heinlein during a research stay at École Polytechnique Fédérale de Lausanne [62]. This verification provides also a second and independent cross-validation of new state of the art expression template-based implementations of these material models due to Paolo Tricerri in `LifeV`, used, e.g., by Tricerri, Dedè, Quarteroni, and Sequeira [133]. This is a nice example, how direct integration of an established high-quality legacy code into a new software framework can increase the software quality of the new implementation through additional cross-validation.

This text is structured as follows: first, Chapter 1 introduces the physiologically realistic geometrical and mechanical models for arterial wall structures based on the polyconvex soft tissue models compared in Brands, Klawonn, Rheinbach, and Schröder [23] and non-overlapping domain decomposition methods, with a focus on the **d**ual **p**rimal **f**inite **e**lement **t**earing and **i**nterconnecting method (FETI-DP). In particular, we shortly discuss the application thereof to nonlinear problems in mechanics as a Newton-Krylov-FETI-DP algorithm. In this context, we also illuminate some easily derivable structural properties of the discretization before we illuminate our load stepping procedures and the tuning thereof. An associated Appendix A is provided to stage, at a gentle pace, the framework of hyperelasticity, the basis for the employed anisotropic, quasi-incompressible and polyconvex soft tissue models. In Chapter 2, we present a new contribution, a multi-language software framework supporting an MPI parallel Newton-Krylov-FETI-DP solver called `mparfeap`. This solver uses a modified version of the `FEAP` distribution developed by Jörg Schröder's research group for the local finite element assembly of the soft-tissue models. The second essential component is a `FETI-DP-solver` developed by Oliver Rheinbach in his dissertation [110]. Further, we present the library interface and aspects of the software engineering and dissect the essential parts of the architecture of the new `FEAP`-interface `libfw`. In the final Chapter 3, we then present a series of applications of the software framework, i.e., of the Newton-Krylov-FETI-DP solver `mparfeap`, to problems in the mechanics of soft biological tissues. More specifically, we present simulation results for the von Mises equivalent stresses which are incurred in a patient-specific arterial wall model by an internal normal pressure in the physiological regime (up to 500 [mmHg]), strong scalability of Newton-Krylov-FETI-DP for these simulations, a series of weak scalability experiments based on simple tension tests and demonstrate the effects of two computationally cheap but, in some cases, highly effective tuning options for load stepping.

We wish our reader a joyful journey into the fruitful and fresh interdisciplinary realm somewhere inbetween the disciplines of numerical mathematics, parallel computing, software engineering, continuum mechanics and the life sciences.

Essen, June 22nd 2014
Andreas Fischle

INRIA (REO, ESTIME), France;
Emory University (Math&CS), U.S.A.

Acknowledgments

The author wishes to acknowledge the use of the Cray XT6m supercomputer at Universität Duisburg-Essen provided by the **C**enter for **C**omputational **S**ciences and **S**imulation (CCSS) and is thankful for the support by the maintenance staff. In particular we acknowledge the exceptional support by René Vergohsen. Further, we acknowledge technical support by Cray Inc. provided by Stephan Anderson. We also acknowledge the use of the software package `Tecplot360` by Tecplot, Inc. for the presented visualizations and the use of `Amira` by FEI for the generation of arterial wall models provided by Martin Lanser.

Contents

CHAPTER 1

Introduction

1.1 Anisotropic Models for Arterial Wall Structures[1]

Quasi-static three-dimensional simulations of an atherosclerotic artery subject to interior blood pressure have been computationally analyzed in detail, e.g., in Balzani, Böse, Brands, Erbel, Klawonn, Rheinbach, and Schröder [14], Brands, Klawonn, Rheinbach, and Schröder [23], Brinkhues, Klawonn, Rheinbach, and Schröder [25], Schröder, Balzani, and Gross [116], Schröder, Klawonn, Balzani, Rheinbach, and Brands [118]; see also Klawonn and Rheinbach [73] for scalability experiments for the FETI-DP method. Of particular interest for our applications is the work Brands, Klawonn, Rheinbach, and Schröder [23] where multiple different models for the physiological material behavior of arteries were analyzed in a comparative study. The latter work is the most direct predecessor of the present one. We use the identical setup regarding the mechanical setting and solution strategy. Moreover, the software framework presented in Chapter 2 reuses the finite implementations in `FEAP` used in Brands, Klawonn, Rheinbach, and Schröder [23]. Our contribution is the transition from the semi-parallel framework there described to the new software infrastructure for massively parallel computations described in Chapter 2. Note that a variety of polyconvex strain energy densities suitable to describe soft biological tissues was proposed in Balzani, Neff, Schröder, and Holzapfel [11] and in the dissertation of Balzani [10].

The present work originates in the research project

Massiv parallele Simulationen von Arterienwänden: Kontinuumsmechanische Modellbildung und numerische Lösung mittels FETI-Gebietszerlegungsverfahren[a]

(KL 2094/1-{1,2}, SCHR 570/7-{1,2})[b]

[a] Massively parallel simulations of arterial walls: Continuum mechanical modeling and numerical solution via FETI domain decomposition methods (title translated by the author).

[b] The project was supported by the Deutsche Forschungsgemeinschaft (DFG) in two funding periods; common grant proposal by PIs Axel Klawonn and Jörg Schröder.

One of the principal objectives of said projects was to advance the degree of realism in the mechanical and geometrical modelling of atherosclerotic patient-specific arterial wall structures. The generation of realistic computational domains from patient-specific **i**ntra**v**ascular **u**ltra**s**ound (IVUS) imaging data is, e.g., a major theme of the dissertation of Brands [20]. An important driving force of research in biomechanics are medical applications. For instance, accurate predictions of the locations in an atherosclerotic artery, where the risk of plaque rupture is high, are of immense interest for the assessment of the risks involved in a surgical intervention. As a contribution Balzani, Böse, Brands, Erbel, Klawonn, Rheinbach, and Schröder [14] reconstructed the 3D geometry for simulations from patient-specific **i**ntravascular **u**ltra**s**ound **v**irtual **h**istology (IVUS-VH) data in combination with angiographic X-ray images. The input data were provided by Dirk Böse and Raimund Erbel; see also Brands, Schröder, Klawonn, Rheinbach, Böse, and Erbel [22] for a condensed presentation.

Patient-specific models are a necessary prerequisite for calculations of stress localizations, indicating high plaque rupture probability for the particular patient. Note that the proposed method is based on in vivo data which is routinely available from the medical diagnosis, which is certainly advantageous in the long term providing for a smooth workflow during risk analysis prior to an intervention. High resolution computations for transmural wall stresses in the physiological regime are presented in Chapter 3 using such patient-specific geometries.

[1] Parts of this introductory section are derived from a text prepared for a D-A-CH research proposal and also from Brands, Klawonn, Rheinbach, and Schröder [23].

The current thesis was written with the focus on scientific software work. We admit, it is outside of our ambition to present a scientific justification of the particular modelling assumptions made. Neither do we broadly discuss the limitations of the mechanical modelling, the most important being that eigenstresses and fluid-structure interaction are not accounted for. Detailed expositions of the mechanical modelling and the assumptions made are available though, and we refer the interested reader to the dissertations by Balzani [10], Brinkhues [24] where the mechanical modelling is presented and developed as a major theme and to Brands [20] for the geometrical modelling. Note also the extensive references provided in said theses. Shorter presentations of the modelling assumptions including also the choice of realistic material parameters are given in Brands, Klawonn, Rheinbach, and Schröder [23], Brinkhues, Klawonn, Rheinbach, and Schröder [25], Klawonn and Rheinbach [73]. A discussion of the results from the perspective of medical research was published in Balzani, Böse, Brands, Erbel, Klawonn, Rheinbach, and Schröder [14]. The latter work illuminates how accurate patient-specific simulations may eventually contribute to the decision making process preceding surgical intervention. Let us stress that the runtime of such simulations is dominated by the time required to solve linear systems arising from finite element discretization. In practice, often 80-90% and even more of the total simulation runtime is spent in the linear solver phases; see also Chapter 3. Thus, fast linear solvers are doubtless a necessity for real-world applications, e.g., in medicine. Robust parallel scalable domain decomposition methods such as FETI-DP or BDDC (see §1.3) can be considered as state of the art methods.

Accurate numerical computations of stress states in arterial walls require a realistic description of the anisotropic nonlinear material behavior of the biological components and also of the problem geometry. Both aspects represent challenging tasks. The soft biological tissues making up the arterial wall structure behave almost incompressibly, undergo large strains, and are characterized by a strong anisotropy. Intensive prior work with respect to the construction and analysis of polyconvex, anisotropic strain energy densities is given in Schröder and Neff [114], and Schröder, Neff, and Balzani [117]. Note that polyconvexity is the key concept in the mathematical existence theory for hyperelasticity formulated by Ball [8, 9]; more specifically, it assures the existence of minimizers for the internal strain energy potential.

In the following subsections we hope to motivate the present work and to gently introduce the reader to the basic physiology of healthy and atherosclerotic arteries, to the proposed mechanical modelling thereof and to give an impression of the geometrical arterial wall models used in Chapter 3. We thus introduce the modelling framework as presented and used in Brands, Klawonn, Rheinbach, and Schröder [23]. An additional, more detailed introduction to the hyperelastic framework relevant to our applications in Chapter 3 is provided in a separate Appendix A together with further references.

1.1.1 The Physiology of Human Arteries

We briefly introduce the physiology of a human artery in order to introduce the considered mechanically relevant parts, their modelling and some terminology used throughout this text.

The structural composition of a healthy human artery is depicted in Figure 1.1 and a microscopy of a stained slide of an elastic artery is displayed in Figure 1.2. For the modelling of the mechanical response of an atheriosclerotic arterial wall, we consider structural models accounting for the following three mechanically relevant parts of the wall structure:

(i) the media (*tunica media*),

(ii) the adventitia (*tunica adventitia* or *tunica externa*), and

(iii) the atherosclerotic plaque components.

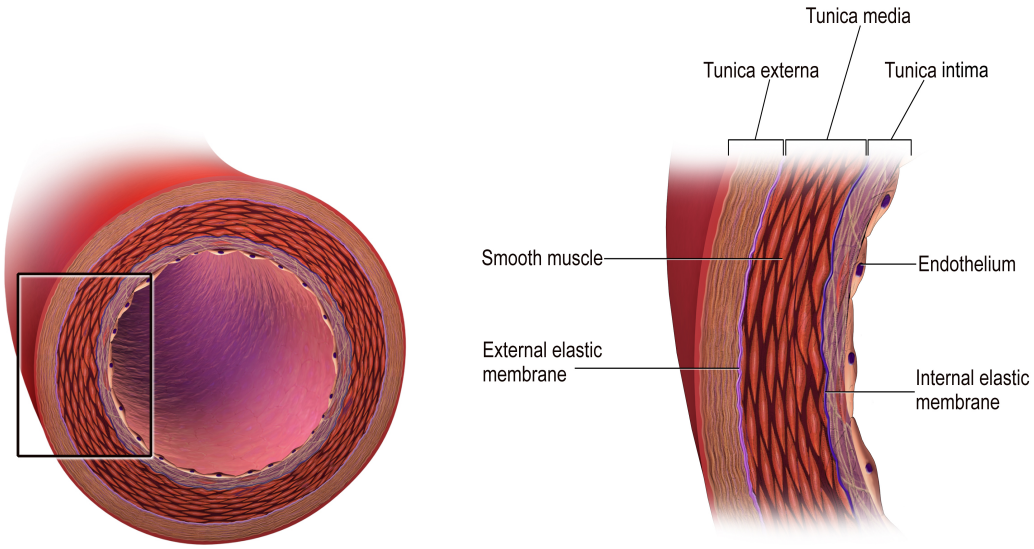


Figure 1.1: Schematic diagrams of a healthy human artery. Original image from Wikimedia Commons, user: BruceBlaus. Blausen Medical - Scientific and Medical Animations `http://blausen.com`

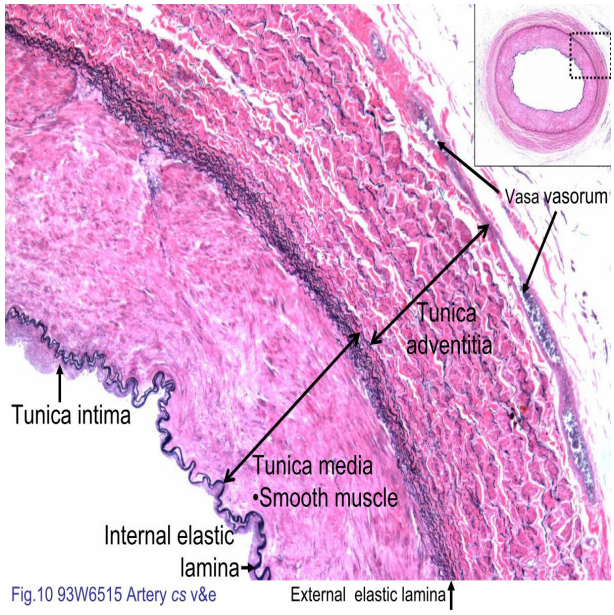


Figure 1.2: A photomicrograph of a slide through an elastic artery. It was stained using Verhoeff's stain and eosin to visualize elastic fibers (black), collagen fibers (pink) and the cell structure. Courtesy of the department of anatomy, Kaohsiung Medical University, Taiwan. Image obtained from `http://anatomy.kmu.edu.tw/BlockHis/Block3/slides/block4_20.html`.

The media and adventitia layers of the wall structure are composed of a ground substance made up of an Elastin fiber network with embedded collagen fibres and smooth muscle cells. The arrangement of these fibers is characterized by two preferred directions $a_i, i = 1, 2$, winding them-

selves helically around the artery; see, e.g., Balzani [10], Brands [20] and Brinkhues [24] for more specific schematic illustrations. In the physiological range of the interior blood pressure, say $\overline{p} \geq 100$ [mmHg], the media and adventitia layers undergo large deformations. Accordingly, they are modeled using the framework of finite anisotropic hyperelasticity. In this framework, the fiber microstructure can be constitutively modeled based on the assumption of a weak interaction between the two fiber families using superimposed transversely isotropic models, see, e.g, Holzapfel, Gasser, and Ogden [64].

The plaque architecture is varying significantly from patient to patient. Its basic components are:

1. fibrotic tissues,
2. calcifications, and
3. extracellular lipid pools.

In the present work, the plaque architecture is simplified. It is represented by a single isotropic constitutive law. More precisely, it is modelled as an isotropic hyperelastic large strain Mooney-Rivlin material, which seems appropriate due to the rather isotropic general nature. Note that the composition of fibers in the fibrotic parts of the plaque is still unclear and subject of investigation.

1.1.2 Mechanical and Geometrical Modelling

We shall take the perspective of the Lagrangian formulation of structural mechanics. In this setting one considers deformation mappings $\varphi : \Omega_{\text{ref}} \to \Omega_{\text{def}}$ which map a reference body $\Omega := \Omega_{\text{ref}}$ into a deformed body Ω_{def}. In quasi-static hyperelasticity, one makes the fundamental assumption of the existence of a functional measuring the internally stored strain energy of a body in a given configuration

$$\varphi \quad \mapsto \quad \Pi^{\text{int}}(\varphi) \, .$$

Equivalently, the displacement field $u = \varphi - \text{id}_\Omega$ may be considered. This is usually preferred in actual implementations. In hyperelasticity, the internal strain energy potential can be expressed as an integral over a local strain energy density depending only on $x \in \Omega$ and the first derivative of the deformation field φ, the so-called **deformation gradient** $F := \nabla\varphi \in \text{GL}^+(3, \mathbb{R})$. Based on this, we introduce the **right Cauchy-Green deformation tensor** $C(F) := F^T F$. A strain energy density can be considered a constitutive law, since it characterizes the local energetic and hence mechanical response of a specific material.

Definition 1.1.1 (Local Strain Energy Density)**.** *A **local strain energy density** is a function*

$$\Psi : \Omega \times \text{GL}^+(3, \mathbb{R}) \to \mathbb{R}_0^+ \, .$$

*It is **homogeneous**, if the dependence on x is only through the deformation gradient $F(x)$.*

Note that the polyconvex energies used in Brands, Klawonn, Rheinbach, and Schröder [23] and in the present work are not homogeneous. They depend on $x \in \Omega$ through two fiber direction fields $a_i : \Omega \to \mathbb{R}^3$, or more precisely, on two associated **structural tensor fields**

$$M^{(i)} \; := \; a_i \otimes a_i \; := \; a_i a_i^T, \quad i = 1, 2 \, .$$

The fiber direction vector fields a_i, $i = 1, 2$ wind themselves helically around the arterial wall layers in the reference configuration. They are hence not constant. See, e.g., the expositions in the dissertation of Balzani [10], and also the dissertation by Brinkhues [24]. For notational convenience, we shall nonetheless drop the dependence of local stored energy densities on $x \in \Omega$ and simply write $\Psi(F)$ for $\Psi(x, F(x))$.

Definition 1.1.2 (Internal Strain Energy)**.** *The* ***internal strain energy*** *of a hyperelastic body* Ω *in an admissible configuration* $\varphi \in \mathcal{A} \subseteq \mathcal{C}^{\text{elas}}_{\Omega}$ *is obtained by integration of the stored strain energy density* $\Psi : \mathrm{GL}^+(3,\mathbb{R}) \to \mathbb{R}^+_0$ *over the reference body*

$$\Pi^{\text{int}} : \mathcal{A} \to \mathbb{R}^+_0, \quad \Pi^{\text{int}}(\varphi) := \int_{\Omega} \Psi(F) \; dV \,. \tag{1.1.1}$$

The term quasi-static refers to the assumption, that the time-derivatives appearing in the dynamic equations of motion can be neglected. The interior virtual work due to the strain is obtained as the first variation of the interior strain energy potential

$$G^{\text{int}} : T\mathcal{A} \to \mathbb{R}, \quad G^{\text{int}}|_{\varphi}(\chi) := \left. \frac{d}{d\varepsilon} \right|_{\varepsilon=0} \Pi^{\text{int}}(\varphi + \varepsilon\chi) \,. \tag{1.1.2}$$

Dirichlet boundary conditions of place are formulated by an appropriate choice of admissible configurations $\mathcal{A}$. Let us introduce the Neumann boundary conditions, i.e., the exterior forces imposed in our applications. We denote by $\partial\Omega_{\text{def}_N} \subseteq \partial\Omega_{\text{def}}$ the Neumann-boundary in the deformed configuration. A traction field $t_N : \partial\Omega_{\text{def}_N} \to \mathbb{R}^3$ is to be imposed there. Further, let $\rho_0 : \Omega \to \mathbb{R}^+$ denote the physical density of the body and let $g : \mathbb{R}^3 \to \mathbb{R}^3$ be a volume force density.

Definition 1.1.3 (Virtual Work due to Exterior Loadings)**.** *We define the* ***external virtual work*** *functional as*

$$\begin{aligned} G^{\text{ext}} : \quad & T\mathcal{A} \to \mathbb{R}, \\ G^{\text{ext}}|_{\varphi}(\chi) \quad := \quad & \underbrace{\int_{\partial\Omega_{\text{def}_N}} \left\langle t_N, \, \chi \circ \varphi^{-1} \right\rangle \; dA}_{\textit{Pressure load}} \quad + \quad \underbrace{\int_{\Omega} \rho_0 \left\langle g, \, \chi \right\rangle \; dV}_{\textit{Volume force}} \,. \end{aligned} \tag{1.1.3}$$

In order to approximate the effects due to the blood pressure inside the arterial wall structure, a pressure load is imposed on the fluid-structure interface. We denote this part of the interface by $\partial\Omega_{\text{def}_N}$ and consider a pressure induced traction field

$$t_N : \partial\Omega_{\text{def}_N} \to \mathbb{R}^3, \qquad\qquad t_N := -\overline{p}\, n \,, \tag{1.1.4}$$

which is normal to the fluid-structure interface. The pressure is assumed to be constant $\overline{p} \geq 0$ [kPa]. Note that the reflected outward unit normal field given by $-n$ determines an inward pointing normal field. Thus, in our applications, a positive blood pressure causes the arterial wall structure to expand. Pressure loadings are configuration dependent loads, a non-trivial type of boundary condition which we shortly discuss in Appendix A. It is well-known that configuration-dependent loads contribute to the finite element stiffness matrix and that this contribution is, depending on the boundary conditions, possibly non-symmetric; see, e.g., the fine monograph by Wriggers [137].

It is generally agreed that the muscle fibers do not respond energetically to compression. The justification is that they can be assumed to buckle instantaneously under compression loads. To incorporate this behavior into the model, we introduce a simple but convenient cut-off function.

Definition 1.1.4 (Macauley Bracket)**.** *The* ***Macauley bracket*** *is given by*

$$\langle \cdot \rangle_m : \mathbb{R} \to \mathbb{R}^+_0, \quad \langle \, x \, \rangle_m := \max\{0, x\} = \begin{cases} x & \textit{for} \quad x \geq 0 \\ 0 \,, & \textit{for} \quad x < 0 \end{cases} . \tag{1.1.5}$$

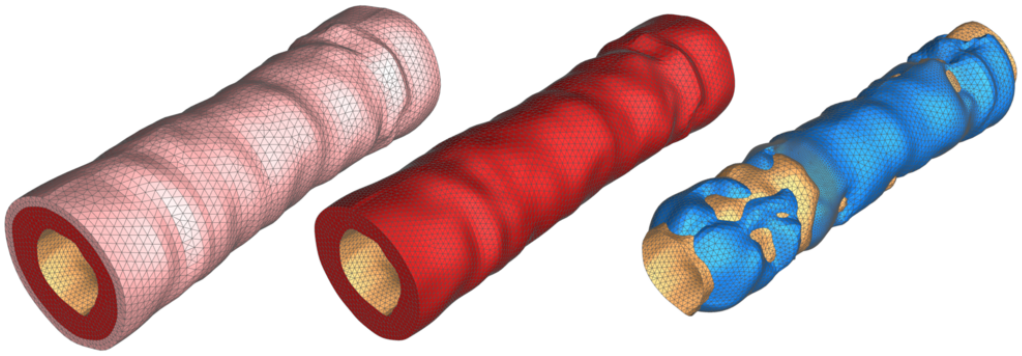


Figure 1.3: Physiological geometry of an artery reconstructed from IVUS imaging data; Image from Balzani, Böse, Brands, Erbel, Klawonn, Rheinbach, and Schröder [14]; see also the preliminary work in Brands, Schröder, Klawonn, Rheinbach, Böse, and Erbel [22]. The outer layers show the geometry of the reconstructed media, intima, and plaque (from left to right).

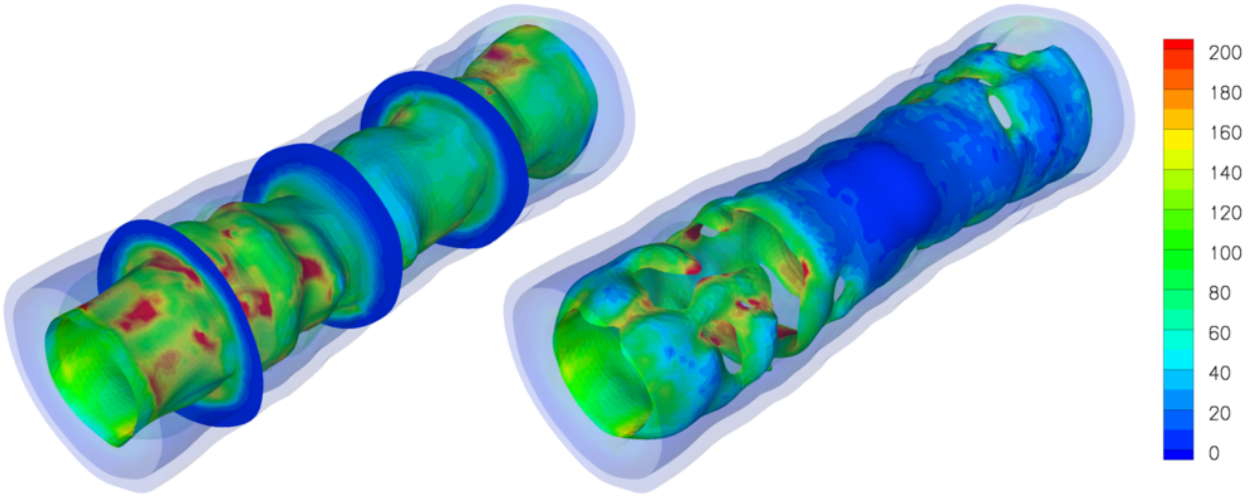


Figure 1.4: Distribution of the von Mises equivalent stresses σ_v due to the strain energy function Ψ_A (media and adventitia layers) and Ψ_{MR} (plaque components). The artery-lumen interface is displayed in three exemplary cross sections (left) and inside the plaque (right). Both show the deformed configuration corresponding to 180 [mmHg] with transparently depicted outer layers. The finite element model has roughly 1.3 million displacement unknowns. Image from Balzani, Böse, Brands, Erbel, Klawonn, Rheinbach, and Schröder [14].

We consider strain energy densities which satisfy the principle of **material frame indifference**, i.e.,

$$\forall Q \in \mathrm{SO}(3): \quad \Psi(QF) = \Psi(F) \,.$$

We introduce the **material symmetry group** $\mathcal{G}$ as the maximal subgroup $\mathcal{G} < \mathrm{SO}(3)$ satisfying

$$\forall Q \in \mathcal{G}: \quad \Psi(FQ) = \Psi(F) \,.$$

If $\mathcal{G} = SO(3)$, $\Psi(F)$ is said to be **isotropic**, otherwise it is an **anisotropic** strain energy density An important special case in the present context are **transversely isotropic** strain energies with preferred direction $a \in \mathbb{R}^3$, i.e.,

$$\exists a \in \mathbb{R}^3: \quad \mathcal{G} \;=\; \mathrm{SO}(3)_a \;:=\; \{Q \in \mathrm{SO}(3) : Qa = a\} \,.$$

Next, we introduce a suitable set of anisotropic polyconvex polynomial invariants using the structural tensors

$$M^{(i)} := a_i \otimes a_i, \quad i = 1, 2 \, .$$

These encode two preferred directions for super-imposed transversal isotropy. In the present context they represent the muscle fiber directions for the adventitia and media layers. In the given setting with two super-imposed preferred directions, we use the following set of polynomial invariants:

Anisotropic Polyconvex Invariants for Two Super-imposed Preferred Directions

Polyconvex Isotropic Invariants:

$$I_1 := \operatorname{tr}[C], \qquad I_2 := \operatorname{tr}[\operatorname{cof}[C]], \quad \text{and} \qquad I_3 := \det[C] \, . \tag{1.1.6}$$

Polyconvex Anisotropic Invariants:

$$J_4^{(i)} := \operatorname{tr}\left[CM^{(i)}\right], \qquad K_3^{(i)} := I_1 J_4^{(i)} - J_5^{(i)}, \qquad i = 1, 2 \, . \tag{1.1.7}$$

Note that the basic mixed anisotropic invariants

$$J_5^{(i)} := \operatorname{tr}\left[C^2 M^{(i)}\right], \quad i = 1, 2 \, ,$$

are not polyconvex; see Merodio and Neff [98]. This is the reason to use $K_3^{(i)}$, $i = 1, 2$, which is polyconvex.

In the following, we introduce the polyconvex, incompressible and anisotropic hyperelastic material models which have been evaluated in Brands, Klawonn, Rheinbach, and Schröder [23]. These are represented in terms of the previously introduced invariants and composed of three basic building blocks:

1. an isotropic contribution due to the ground substance matrix,
2. a penalty term for volume change and
3. an anisotropic fiber contribution generated by two-superimposed transversely isotropic parts.

The fiber contributions model the response due to the muscle fibers in the adventitia and media layers and the volumetric penalty term leads to a quasi-incompressible material behavior. The material models available in `mparfeap` are accessed through the `FEAP` interface `libfw-js` described in Chapter 2. We have compiled the models used for Brands, Klawonn, Rheinbach, and Schröder [23] in our Table 1.1.

For the numerical applications presented in Chapter 3, we have decided to concentrate our attention exclusively on two strain energy densities:[2]

- Ψ_A is used for high-resolution computations of von Mises equivalent stresses in §3.2 to assess the strong scalability thereof in §3.3 and to study the effect of two simple load stepping tuning options in §3.5;
- Ψ_B is used in a series of simple tension tests and we investigate the weak scalability in §3.4 and the effects of load stepping tuning in §3.5.

[2] Note that the other three energy densities Ψ_C, Ψ_D and Ψ_E are available in `libfw-js`, i.e., they can be flexibly used without restrictions.

We have compiled the second Piola-Kirchhoff stress tensors for the strain energies listed in Table 1.1 in a separate Table 1.2. Let us emphasize that *the software component `libfw-js` presented in Chapter 2 allows to reuse the implementations of the material models in `FEAP`, used, e.g., in Brands, Klawonn, Rheinbach, and Schröder [23].* To be precise, some minor additional modifications have been introduced by the present author, in particular a more refined error handling. These modifications support the implementation of a failure-based adaptive load stepping strategy in `mparfeap`.

The derivation of the stress tensors compiled in Table 1.2 is based on the relation $S_2 = 2\,D_C\Psi(C)$, the chain rule and formulas for the matrix representations of the derivatives of the invariants; see, e.g., the appendix to Brinkhues [24] including also the second derivatives of the invariants.

Derivatives of the Polyconvex Invariants

Isotropic principal invariants:

$$D_C I_1(C) = \mathbb{1}, \quad D_C I_2(C) = I_1(C)\,\mathbb{1} - C, \quad \text{and} \quad D_C I_3(C) = I_3(C) C^{-1} \,. \tag{1.1.8}$$

Anisotropic main invariants:

$$D_C J_4(C,M) = M, \quad D_C J_5(C,M) = CM^T + M^T C = 2\,\mathrm{sym}\,[CM] \,. \tag{1.1.9}$$

Modified polyconvex invariant:

$$\begin{aligned} D_C K_3(C,M) &= D_C\Big(I_1(C) J_4(C,M) - J_5(C,M)\Big) \\ &= J_4(C,M)\,\mathbb{1} + I_1(C) M - 2\,\mathrm{sym}\,[CM] \,. \end{aligned} \tag{1.1.10}$$

As in Brands, Klawonn, Rheinbach, and Schröder [23], we model the plaque components as an isotropic Mooney-Rivlin solid. We use exactly the same formulation and implementation on which the comparative studies in Brands, Klawonn, Rheinbach, and Schröder [23] were based. We have compiled them in Table 1.1.

Isotropic Mooney-Rivlin Model for the Plaque Components (cf. [23])

Stored Energy Density:

$$\Psi_{\mathrm{MR}}(C) = \beta_1 I_1 + \eta_1 I_2 + \delta_1 I_3 - \delta_2 \log(I_3) \,. \tag{1.1.11}$$

Second Piola Kirchhoff Stress Tensor:

$$\frac{1}{2} S_{2,\mathrm{RM}} = \beta_1\,\mathbb{1} + \eta_1\,(I_1\,\mathbb{1} - C) + \delta_1 I_3 C^{-1} - \delta_2 C^{-1} \,. \tag{1.1.12}$$

For the analysis of the transmural stress distribution inside an arterial wall, the Cauchy stresses for the deformed configuration are computed from the second Piola-Kirchhoff tensor via the Piola transformation given by

$$\sigma = J^{-1} F S_2 F^T \,.$$

From this, the von Mises equivalent stress σ_v, also called the effective stress, is determined. It is a scalar stress quantity and is commonly interpreted as a failure indicator. It is quite convenient for visualization purposes. The main principle on which the derivation of the von Mises equivalent stresses is based, is that material failure is only caused by shear stresses, i.e., it is not influenced by dilatatory stress components.

Polyconvex Soft-Tissue Models for the Media and Adventitia Layers (cf. [23])

Model $\Psi_{\mathbf{A}}$: (Balzani, Neff, Schröder, and Holzapfel [11])

$$\Psi_A\left(C, M^{(1)}, M^{(2)}\right) = \underbrace{c_1\left(\frac{I_1}{I_3^{1/3}} - 3\right)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\left(I_3^{\varepsilon_2} + \frac{1}{I_3^{\varepsilon_2}} - 2\right)}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2} \alpha_1 \left\langle K_3^{(i)} - 2\right\rangle_m^{\alpha_2}}_{\text{Fiber contributions}} .$$

Model $\Psi_{\mathbf{B}}$: (Holzapfel, Gasser, and Ogden [64])

$$\Psi_B\left(C, M^{(1)}, M^{(2)}\right) = \underbrace{c_1\left(\frac{I_1}{I_3^{1/3}} - 3\right)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\left(I_3^{\varepsilon_2} + \frac{1}{I_3^{\varepsilon_2}} - 2\right)^{\alpha_5}}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2} \frac{k_1}{2k_2}\left\{\exp\left(k_2\left\langle \frac{J_4^{(i)}}{I_3^{1/3}} - 1\right\rangle_m^2\right) - 1\right\}}_{\text{Fiber contributions}} .$$

Model $\Psi_{\mathbf{C}}$: (Balzani, Neff, Schröder, and Holzapfel [11])

$$\Psi_C\left(C, M^{(1)}, M^{(2)}\right) = \underbrace{c_1\left(\frac{I_1}{I_3^{1/3}} - 3\right)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\left(I_3^{\varepsilon_2} + \frac{1}{I_3^{\varepsilon_2}} - 2\right)}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2} \alpha_3 \left\langle \frac{J_4^{(i)}}{I_3^{1/3}} - 1\right\rangle_m^{\alpha_4}}_{\text{Fiber contributions}} .$$

Model $\Psi_{\mathbf{D}}$: (Balzani, Neff, Schröder, and Holzapfel [11])

$$\Psi_D\left(C, M^{(1)}, M^{(2)}\right) = \underbrace{c_1\Big(I_1 - \log(I_3)\Big)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\left(I_3^{\varepsilon_2} + \frac{1}{I_3^{\varepsilon_2}} - 2\right)}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2} \alpha_3 \left\langle J_4^{(i)} - 1\right\rangle_m^{\alpha_4}}_{\text{Fiber contributions}} .$$

Model $\Psi_{\mathbf{E}}$: (Holzapfel, Gasser, and Ogden [65])

$$\Psi_E\left(C, M^{(1)}, M^{(2)}\right) = \underbrace{c_1\Big(I_1 - \log(I_3)\Big)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\left(I_3^{\varepsilon_2} + \frac{1}{I_3^{\varepsilon_2}} - 2\right)}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2} \frac{k_1}{2k_2}\left\{\exp\left(k_2\left\langle J_4^{(i)} - 1\right\rangle_m^2\right) - 1\right\}}_{\text{Fiber contributions}} .$$

Table 1.1: The polyconvex anisotropic and quasi-incompressible strain energy densities were used in Brands, Klawonn, Rheinbach, and Schröder [23]. They are available in `FEAP-JS` and accessible to `mparfeap` via the `libfw-js` wrapper.

Definition 1.1.5 (Von Mises Equivalent Stresses)**.** *Let σ denote the Cauchy stress tensor in a deformed admissible configuration $\varphi \in \mathcal{A}$. The* ***von Mises equivalent stresses*** *σ_v are given by*

$$\sigma_v^2 = \frac{3}{2}\left\|\operatorname{dev}[\sigma]\right\|^2 = \frac{3}{2}\left\|\sigma - \frac{1}{3}\operatorname{tr}[\sigma]\cdot\mathbb{1}\right\|^2 \tag{1.1.13}$$

$$= \frac{(\sigma_{11}-\sigma_{22})^2 + (\sigma_{22}-\sigma_{33})^2 + (\sigma_{33}-\sigma_{11})^2 + 6(\sigma_{12}^2 + \sigma_{23}^2 + \sigma_{13}^2)}{2} \tag{1.1.14}$$

Second Piola Kirchhoff Stress Tensors (cf. [23])

Model $\Psi_{\mathbf{A}}$:

$$\frac{1}{2}S_{2,A} = \underbrace{\frac{c_1}{I_3^{1/3}}\left(\mathbb{1} - \frac{I_1}{3}C^{-1}\right)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\varepsilon_2\left(I_3^{\varepsilon_2} - \frac{1}{I_3^{\varepsilon_2}}\right)C^{-1}}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2}\alpha_1\alpha_2\left\langle K_3^{(i)} - 2\right\rangle_m^{\alpha_2-1}\left(J_4^{(i)}\,\mathbb{1} + I_1\,M^{(i)} - 2\,\mathrm{sym}\left[CM^{(i)}\right]\right)}_{\text{Fiber contributions}}.$$

Model $\Psi_{\mathbf{B}}$:

$$\frac{1}{2}S_{2,B} = \underbrace{\frac{c_1}{I_3^{1/3}}\left(\mathbb{1} - \frac{I_1}{3}C^{-1}\right)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\varepsilon_2\alpha_5\left(I_3^{\varepsilon_2} + \frac{1}{I_3^{\varepsilon_2}} - 2\right)^{\alpha_5-1}\left(I_3^{\varepsilon_2} - \frac{1}{I_3^{\varepsilon_2}}\right)C^{-1}}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2}\frac{k_1}{I_3^{1/3}}\exp\left(k_2\left\langle\frac{J_4^{(i)}}{I_3^{1/3}} - 1\right\rangle_m^2\right)\left\langle\frac{J_4^{(i)}}{I_3^{1/3}} - 1\right\rangle_m\left(M^{(i)} - \frac{1}{3}C^{-1}\right)}_{\text{Fiber contributions}}.$$

Model $\Psi_{\mathbf{C}}$:

$$\frac{1}{2}S_{2,C} = \underbrace{\frac{c_1}{I_3^{1/3}}\left(\mathbb{1} - \frac{I_1}{3}C^{-1}\right)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\varepsilon_2\left(I_3^{\varepsilon_2} - \frac{1}{I_3^{\varepsilon_2}}\right)C^{-1}}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2}\frac{\alpha_3\alpha_4}{I_3^{1/3}}\left\langle\frac{J_4^{(i)}}{I_3^{1/3}} - 1\right\rangle_m^{\alpha_4-1}\left(M^{(i)} - \frac{1}{3}C^{-1}\right)}_{\text{Fiber contributions}}.$$

Model $\Psi_{\mathbf{D}}$:

$$\frac{1}{2}S_{2,D} = \underbrace{c_1\left(\mathbb{1} - C^{-1}\right)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\varepsilon_2\left(I_3^{\varepsilon_2} - \frac{1}{I_3^{\varepsilon_2}}\right)C^{-1}}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2}\alpha_3\alpha_4\left\langle J_4^{(i)} - 1\right\rangle_m^{\alpha_4-1}M^{(i)}}_{\text{Fiber contributions}}.$$

Model $\Psi_{\mathbf{E}}$:

$$\frac{1}{2}S_{2,E} = \underbrace{c_1\left(\mathbb{1} - C^{-1}\right)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\varepsilon_2\left(I_3^{\varepsilon_2} - \frac{1}{I_3^{\varepsilon_2}}\right)C^{-1}}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2}k_1\exp\left(k_2\left\langle J_4^{(i)} - 1\right\rangle_m^2\right)\left\langle J_4^{(i)} - 1\right\rangle_m M^{(i)}}_{\text{Fiber contributions}}.$$

Table 1.2: Second Piola-Kirchhoff stress tensors for the polyconvex anisotropic and quasi-incompressible strain energy densities used in Brands, Klawonn, Rheinbach, and Schröder [23].

$$= \sigma_{11}^2 + \sigma_{22}^2 + \sigma_{33}^2 - (\sigma_{11}\sigma_{22} + \sigma_{22}\sigma_{33} + \sigma_{33}\sigma_{11}) + 3(\sigma_{12}^2 + \sigma_{23}^2 + \sigma_{31}^2) \, . \tag{1.1.15}$$

1.1.3 Material Parameter Sets Fitted to Experimental Data

In this subsection, we reproduce and shortly discuss the material parameter sets for the previously presented stored energy functions $\Psi_X, X = A, B, C, D, E$, exactly as they were introduced in Brands, Klawonn, Rheinbach, and Schröder [23]. Also in Brands, Klawonn, Rheinbach, and Schröder [23], a description of the fitting to experimentally measured data is given; see also Balzani [10] for prior work. Further improvements to this process are given in Brinkhues, Klawonn, Rheinbach, and Schröder [25] and Brinkhues [24].

The presented simulations in Chapter 3 are based on the material parameter sets compiled in Table 1.3.

Material Parameters for the Adventitita and Media Layers (cf. [23])

Set	Model	Layer	$\mathbf{c_1}$ [kPa]	ε_1 [kPa]	ε_2 [-]	α_1 [kPa]	α_2 [-]	α_3 [kPa]	α_4 [-]	α_5 [-]	$\mathbf{k_1}$ [kPa]	$\mathbf{k_2}$ [-]
1	Ψ_A	adv.	7.5	100.0	20.0	$1.5 \cdot 10^{10}$	20.0	–	–	–	–	–
		med.	17.5	100.0	50.0	$5.0 \cdot 10^{5}$	7.0	–	–	–	–	–
2	Ψ_A	adv.	6.6	23.9	10.0	1503.0	6.3	–	–	–	–	–
		med.	17.5	499.8	2.4	30001.9	5.1	–	–	–	–	–
3	Ψ_B	adv.	6.2	101.0	10.0	–	–	–	–	3.0	6.0	20.0
		med.	10.7	207.1	9.7	–	–	–	–	10.0	1018.8	20.0
4	Ψ_C	adv.	6.8	52.5	10.0	–	–	1005.2	6.3	–	–	–
		med.	11.8	84.2	5.94	–	–	49999.1	4.1	–	–	–
5	Ψ_D	adv.	7.0	50.3	3.0	–	–	10010.8	8.2	–	–	–
		med.	10.9	84.4	5.69	–	–	10002.1	3.32	–	–	–
6	Ψ_E	adv.	6.1	49.7	3.1	–	–	–	–	–	5.5	20.0
		med.	9.7	95.3	3.8	–	–	–	–	–	687.6	20.0

Table 1.3: Sets of material parameters for the representation of adventitia (adv.) and media (med.) layers. Set-1 is taken from Balzani [10]. All other parameter sets were computed using an optimization strategy based on least squares fittings. The exact procedure is described in Brands, Klawonn, Rheinbach, and Schröder [23] and was improved in Brinkhues, Klawonn, Rheinbach, and Schröder [25].

Note that the penalty parameters for the volume penalty term have also been fitted to experimental data. Further, some principal restrictions on the material parameters apply:

$$\begin{aligned} &c_1 > 0, \quad \varepsilon_1 > 0, \quad \varepsilon_2 > 1, \quad \alpha_1 > 0, \quad \alpha_2 > 2 \, , \\ &\alpha_3 > 0, \quad \alpha_4 > 2, \quad \alpha_5 > 2, \quad k_1 > 0, \quad k_2 > 0 \, . \end{aligned} \tag{1.1.16}$$

The penalty weight factor ε_1 is intuitively required to be positive. For the volume penalty function ε_2 is restricted so that the penalty function is convex in $I_3 = \det[C]$. For $\alpha_4 > 2$ and $\alpha_5 > 2$, the stored energy densities can be seen to be twice continuously differentiable. This is a necessary prerequisite for Fréchet differentiability of the strain energy potential.

Remark 1.1.6 (Relaxed Volume Penalty Parameters)**.** *For some of our computations presented in Chapter 3, we have relaxed the volume penalty parameters by setting $\varepsilon_1 = 10$ and $\varepsilon_2 = 4$. This alteration to a material parameter set is marked by the keyword* ***relaxed*** *next to the number of the parameter set, e.g., Set-2 relaxed.*

For similar choices of parameters in the context of Augmented Lagrange methods, see Brinkhues, Klawonn, Rheinbach, and Schröder [25].

Precisely following the computations presented in Brands, Klawonn, Rheinbach, and Schröder [23], we use the same set of material parameters for the plaque components in the present work. More precisely, the plaque architecture is modeled by the strain energy density Ψ_{MR}.

Material Parameters for the Plaque Components

$$\beta_1 = 80 \text{ [kPa]} \qquad \eta_1 = 250 \text{ [kPa]} , \qquad \text{and} \qquad \delta_1 = 2000 \text{ [kPa]} .$$

In order to obtain a stress-free reference state for the plaque material model Ψ_{MR}, the material parameter δ_2 is determined by the equation

$$0 \;=\; \beta_1 + 2\eta_1 + \delta_1 - \delta_2 \; . \tag{1.1.17}$$

This is implied by the particular form of the presented second Piola Kirchhoff stress tensor $S_{2,\mathrm{MR}}$ for $C = \mathbb{1}$.

1.2 Aspects of Nonlinear Finite Element Methods

We shall focus on selected aspects of nonlinear finite element methods which are of particular interest for the present thesis. These are applied to the previously presented anisotropic hyperelastic framework. For a detailed exposition of the implementation of the polyconvex, anisotropic and incompressible material laws considered here, we refer the interested reader to Balzani [10] and to Brinkhues [24]. In the present context, we have explicitly strived to use `libfw-js` as a blackbox material library with a well-defined interface. For a general introduction to nonlinear finite element methods, see, e.g., Ciarlet [28, 29] or Wriggers [137].

For the introduction of finite element methods, we choose a configuration space for the body given by the following weakly differentiable Sobolev functions

$$\mathcal{C}_\Omega = \{u \in H^1(\Omega)^3 : \det[\nabla u] > 0\} \; .$$

The admissible configurations are to satisfy prescribed Dirichlet boundary conditions given by $u_D \in H^{1/2}(\partial\Omega)$, i.e.,

$$\mathcal{A} = \{u \in \mathcal{C}_\Omega : u|_{\partial\Omega_D} = u_D\} \; .$$

Admissible variations are required to leave the Dirichlet boundary conditions fixed, i.e., we have

$$T_\varphi\mathcal{A} = \{\Delta u \in H^1(\Omega)^3 : \Delta u|_{\partial\Omega_D} = 0\} \; .$$

Note that $T_\varphi\mathcal{A}$ corresponds to the space of Newton corrections in Newton's method. This implies that the values on the Dirichlet boundary are never updated in Newton's algorithm, i.e., we have

$$\forall n \geq 0 : \quad u^{(n+1)}|_{\Omega_D} \;=\; u^{(n)}|_{\Omega_D} + \underbrace{\Delta u^{(n)}|_{\Omega_D}}_{=\,0} \;=\; u^{(0)}|_{\Omega_D} = u_D \; .$$

This sets the stage for the introduction of the discretization and linearization of the balance of virtual work using nonlinear finite element methods.

1.2.1 Linearization of the Equilibrium Equations

In the given frame of finite anisotropic hyperelasticity, the problem at hand is to numerically approximate an admissible configuration of the problem geometry Ω in equilibrium satisfying the boundary conditions of place and tractions. In what follows, we derive the general form of the linear problems to be solved using parallel FETI-DP methods. These are introduced in §1.3 and represent the linear solution strategy for the Newton-Krylov-FETI-DP solver `mparfeap` presented in Chapter 2. This solver was used for the numerical applications presented in Chapter 3.

Equilibrium of Forces

A hyperelastic body $(\Omega, \mathcal{C}^{\text{elas}}_{\Omega}, \Psi)$ in configuration $\varphi \in \mathcal{A} \subseteq \mathcal{C}^{\text{elas}}_{\Omega}$ is in equilibrium if

$$\forall \delta\varphi \in T_{\varphi}\mathcal{A} \quad : \quad G^{\text{int}}|_{\varphi}(\delta\varphi) \; - \; G^{\text{ext}}|_{\varphi}(\delta\varphi) = 0 \, . \tag{1.2.18}$$

Let us now briefly indicate how this problem is linearized and discretized using finite element methods for the application of a Newton-Krylov-FETI-DP method. We introduce a finite element subspace

$$V^h \; := \; \text{span}\left(\left\{(e^h_i)_{1\leq i\leq N^h}\right\}\right) \; \subseteq \; T_{\varphi}\mathcal{A} \, .$$

It is spanned by basis functions $(e^h_i)_{1\leq i\leq N^h}$ and is of dimension $\dim V^h = N^h$.

The globally assembled stiffness matrices and load vectors are obtained by inserting the finite element basis functions into the linearizations of the internal and external virtual work functionals:

$$K^{\text{int}}(\varphi^h)_{ij} \; := \; d_{\varphi}G^{\text{int}}\big|_{\varphi^h}(e^h_i, e^h_j) \; = \; d^2_{\varphi}\Pi^{\text{int}}\big|_{\varphi^h}(e^h_i, e^h_j) \qquad \text{for} \quad 1 \leq i,j \leq N^h \, , \tag{1.2.19}$$

$$K^{\text{ext}}(\varphi^h)_{ij} \; := \; d_{\varphi}G^{\text{ext}}\big|_{\varphi^h}(e^h_i, e^h_j) \qquad \text{for} \quad 1 \leq i,j \leq N^h \, , \tag{1.2.20}$$

$$f^{\text{int}}(\varphi^h)_i \; := \; -\, G^{\text{int}}\big|_{\varphi^h}(e^h_i) \qquad \text{for} \quad 1 \leq i \leq N^h \, , \tag{1.2.21}$$

$$f^{\text{ext}}(\varphi^h)_i \; := \; -\, G^{\text{ext}}\big|_{\varphi^h}(e^h_i) \qquad \text{for} \quad 1 \leq i \leq N^h \, . \tag{1.2.22}$$

For notational convenience, we shall identify finite element functions with their coefficient vectors, e.g., we identify

$$[\delta\varphi^h] \; \in \mathbb{R}^{N^h} \cong \; \delta\varphi^h = \sum_{i=1}^{N^h} [\delta\varphi^h]_i \, e^h_i \; \in V^h \, , \text{ and} \tag{1.2.23}$$

$$[\Delta\varphi^h] \; \in \mathbb{R}^{N^h} \cong \Delta\varphi^h = \sum_{j=1}^{N^h} [\Delta\varphi^h]_j \, e^h_j \; \in V^h \, . \tag{1.2.24}$$

Let us now work towards a first order Taylor expansion of the equilibrium equation based at φ^h for a given perturbation $\Delta\varphi^h$. We first expand a virtual work functional in general form to first order

$$G|_{\varphi^h+\Delta\varphi^h}(\delta\varphi^h) = G|_{\varphi^h}(\delta\varphi^h) + d_{\varphi}G|_{\varphi^h}(\delta\varphi^h, \Delta\varphi^h) \qquad + \text{ h. o. t. in } \Delta\varphi^h \tag{1.2.25}$$

$$= -(\delta\varphi^h)^T f(\varphi^h) + (\delta\varphi^h)^T K(\varphi^h)\Delta\varphi^h \qquad + \text{ h. o. t. in } \Delta\varphi^h \tag{1.2.26}$$

$$= (\delta\varphi^h)^T \Big[K(\varphi^h)\Delta\varphi^h - f(\varphi^h)\Big] \qquad + \text{ h. o. t. in } \Delta\varphi^h \, . \tag{1.2.27}$$

Applying this to both G^{int} and G^{ext} separately and dropping the higher order terms, we arrive at the discretization of the linearized condition for an equilibrium of virtual work

$$\forall \delta\varphi^h \in V^h : \quad 0 \; = \; (\delta\varphi^h)^T \Bigg[\underbrace{\Big(K^{\text{int}}(\varphi^h) - K^{\text{ext}}(\varphi^h)\Big)}_{=:\, K(\varphi^h)} \Delta\varphi^h - \underbrace{\Big(f^{\text{int}}(\varphi^h) - f^{\text{ext}}(\varphi^h)\Big)}_{=:\, f(\varphi^h)} \Bigg] \, . \tag{1.2.28}$$

Assuming invertibility, this is satisfied by a Newton-correction $\Delta\varphi^h$ if and only if it solves the bracketed linear equation system

$$\left[\underbrace{K^{\text{int}}(\varphi^h)}_{\text{symmetric}} \quad - \quad \underbrace{K^{\text{ext}}(\varphi^h)}_{\substack{\text{possibly} \\ \text{non-symmetric}}} \right] \Delta\varphi^h \;=\; f^{\text{int}}(\varphi^h) \;-\; f^{\text{ext}}(\varphi^h) \,. \tag{1.2.29}$$

Thus, to determine a Newton-correction for φ, in general, we have to solve a *possibly* non-symmetric linear equation system

$$K(\varphi^h)\Delta\varphi^h \quad = \quad f(\varphi^h) \tag{1.2.30}$$

in every step of Newton's algorithm. An equivalent formulation written with respect to the displacement $u := \varphi - \mathrm{id}_\Omega$ is straight-forward to derive. In the present work, we shall use a parallel FETI-DP method for the computation of the Newton corrections. Note that *we have used a symmetric linear solver* for all presented computations. To this end, the `FETI-DP-solver` due to Rheinbach [110] was initialized from the upper right triangle of $K(\varphi^h)$. In other words, this strategy uses the symmetric system matrix with entries

$$K^S(\varphi^h)_{ij} \quad := \quad \begin{cases} K(\varphi^h)_{ij} & : \quad \text{for} \quad i \leq j \\ K(\varphi^h)_{ji} & : \quad \text{for} \quad i > j \,. \end{cases} \tag{1.2.31}$$

For pressure loads leading to a non-symmetric contribution $K^{\text{ext}}(\varphi^h)$, we have that $K(\varphi^h) \neq K^S(\varphi^h)$. For computations without pressure loads (or with conservative pressure loads), it follows that $K(\varphi^h) = K^S(\varphi^h)$. The latter is the case for our weak-scalability experiments for biological soft tissues presented in §3.4. Note that these tension tests are formulated using Dirichlet boundary data only. In our computations of the von Mises equivalent stresses in §3.2, the Newton-Krylov-FETI-DP solver computed the Newton corrections determined by the system

$$K^S(\varphi^h)\Delta\varphi^h \quad = \quad f(\varphi^h) \,. \tag{1.2.32}$$

There is a large body of literature on so-called Newton-like methods with modified tangent stiffness matrices. The use of approximate tangents is well-known to affect the convergence rate of Newton's method. Although a deterioration is in general to be expected, we have not noticed any adverse effects, i.e., we have observed a quadratic convergence rate in our numerical experiments in Chapter 3. This may be due to a mild non-symmetry. In such cases, the use of $K^S(\varphi^h)$ or $\mathrm{sym}\left[K(\varphi^h)\right] = \frac{1}{2}(K(\varphi^h) + K(\varphi^h)^T)$ as the system matrix is an attractive approach, often used in the engineering community. Similar to the present approach, non-symmetric tangent stiffnesses were symmetrized in Pierson [107, p. 64]. There, probably, the author used system matrices of the form $\mathrm{sym}\left[K(\varphi^h)\right] = \frac{1}{2}(K(\varphi^h) + K(\varphi^h)^T)$ for the application of Newton-Krylov-FETI-DP to co-rotational nonlinear elasticity. Note that, also in Pierson [107, p. 64], a reference is given to the dissertation by Haugen [61] indicating that the use of a symmetrized tangent has been observed to not deteriorate the convergence properties of the outer Newton's method. On a sidenote, we also want to mention an interesting account given by Simo [121] on the symmetry of second derivatives.

1.2.2 Symmetry and Spectrum of the Finite Element Stiffness Matrix

Let us shortly present what is well-known regarding the symmetry of the discretized stiffness matrices and on their spectrum. For the construction of suitable preconditioners, it is certainly of interest under which conditions the finite element stiffness matrices arising from the linearization of the equilibrium equations of nonlinear hyperelasticity are symmetric for a standard Galerkin-ansatz. We recall that the finite element matrix in our setting consists of two contributions:

$$K(\varphi^h) := K^{\text{int}}(\varphi^h) - K^{\text{ext}}(\varphi^h) \,.$$

It is not hard to see the symmetry of the bilinear form obtained from the linearization of the *interior* virtual work functional G^{int}. Inserting a pair of finite element basis functions $e_i^h, e_j^h \in V_h \subseteq T_\varphi \mathcal{A}$, we easily infer symmetry of the stiffness matrix

$$K^{\text{int}}(\varphi^h)_{ij} \;=\; d^2_\varphi \Pi^{\text{int}}|_{\varphi^h}(e_i^h, e_j^h) \;=\; d^2_\varphi \Pi^{\text{int}}|_{\varphi^h}(e_j^h, e_i^h) \;=\; K^{\text{int}}(\varphi^h)_{ji} \,. \tag{1.2.33}$$

Detailed derivations for the first and second variation of the internal strain energy in a displacement formulation are given in Appendix A. They are, however, not derived for the $\overline{F}$-method which we have used to reduce volume locking effects and a remark seems in order.

Remark 1.2.1 (Symmetry of $K^{\text{int}}(\varphi)$ in $\overline{\text{F}}$ methods)**.** *Following the dissertation by Freischläger [54], it is known that the symmetry of the global finite element tangent stiffness matrix $K^{\text{int}}(\varphi)$ is not affected by the $\overline{\text{F}}$-method used in Brands, Klawonn, Rheinbach, and Schröder [23]. That method is based on Simo [122]; see also Nagtegaal and Fox [99]. Note that we have used the same implementation for the computations presented in the present thesis. This is not necessarily the case, e.g., Souza and Neto proposed a method which, in general, leads to a non-symmetric stiffness matrix $K^{\text{int}}(\varphi^h)$.*

If the prescribed configuration-dependent loadings are not conservative, the matrix $K^{\text{ext}}(\varphi^h)$ is not symmetric. Any information on the structure of the spectrum and on positive definiteness of the contributions to $K(\varphi^h)$ are certainly of interest. For example, the implementation of finite element approximations to material laws in hyperelastic formulations is a demanding task and thus error prone. A possible, very simple sanity check is to verify the symmetry of the interior tangent stiffness matrix $K^{\text{int}}(\varphi^h)$. Note that for a displacement (or an $\overline{\text{F}}$-formulation) with dead loads, or conservative configuration-dependent loadings, the matrix $K(\varphi^h)$ is also symmetric.

Let us turn towards the spectrum of the stiffness matrices. Knowledge on the spectrum of the finite element stiffness matrices $K^{\text{int}}(\varphi^h)$ is certainly of interest for the construction of preconditioners. The study of the spectrum of the finite element stiffness matrix is routine in stability and bifurcation analysis of structures, see, e.g. the monographs due to Allen and Bulson [2], Bažant and Cedolin [16], Thompson and Hunt [131]; see also Wriggers [137] for a very nice condensed overview.

Since $K^{\text{int}}(\varphi^h)$ is a real symmetric matrix, it has real spectrum

$$\text{spec}(K^{\text{int}}(\varphi^h)) \;\subseteq\; \mathbb{R} \,.$$

In particular, the spectrum may contain negative eigenvalues. Further, for non-conservative configuration-dependent loads, i.e., loadingss which do not derive from a potential, the symmetry of the stiffness matrix $K(\varphi^h)$ is in general lost, since $K^{\text{ext}}(\varphi^h)$ is not symmetric then. This just implies the obvious

$$\text{spec}(K(\varphi^h)) \;\subseteq\; \mathbb{C}, \quad \text{while} \quad \text{spec}(K^S(\varphi^h)) \;\subseteq\; \mathbb{R}$$

which is, it seems, all that can be expected, in general.

Note that unstable equilibria such as buckling and other types of elastic instability are a physical reality. In the discretized setting, these appear as negative or zero eigenvalues contained in $\text{spec}(K(\varphi^h))$. Polyconvex strain energy densities do allow for unstable equilibria; see Balzani [10, p. 50]. In other words, indefinite finite element system matrices $K^{\text{int}}(\varphi^h)$ may arise and may be unexpected for polyconvexity is a generalization of convexity. The possible indefiniteness of $K^{\text{int}}(\varphi^h)$ has to be seen as a necessary requirement of the framework.

Currently, there is no support for large scale stability analysis in the presented Newton-Krylov-FETI-DP solver `mparfeap`. In `mparfeap`, we have enabled the use of symmetric storage and solvers in the `PETSc`-based `FETI-DP-solver` due to Rheinbach [110]. This reduces the critical

memory footprint of the linear solution phases. In turn, the parallel scalability of Newton-Krylov-FETI-DP was improved, as this allows for larger coarse problems. More precisely, it allows for the Cholesky-factorization thereof.

1.2.3 Load Stepping as a Homotopy Method

In computational structural mechanics, it is common practice to consider homotopy methods parametrizing the boundary conditions. This is called load-stepping or incremental loading. Typically the surface loadings or exterior forces to which a given mechanical structure is subjected cannot be imposed instantaneously since the nonlinear problem at hand is too ill-conditioned for this to succeed. It is natural to load the structure incrementally, considering the time-dependent case. In the dynamic equations of motion, the structure is loaded with a loading rate, i.e., the full load is never instantaneously imposed in reality.

In the previously presented quasi-static setting for hyperelasticity, we have to compute a root $u^* \in V^h \subseteq \mathcal{A}$ of the equilibrium equation. For this, we shall use an adaptive load stepping strategy implemented in `mparfeap`. The classical load stepping method is a particular homotopy method. We consider it based on the well-known Newton's method. Let us introduce the **nonlinear residual** for a discrete displacement $\mathcal{R} : \mathbb{R}^n \to \mathbb{R}^n$ and the associated root finding problem

$$\mathcal{R}(u^*) \stackrel{!}{=} 0 \, .$$

If $\mathcal{R}$ is differentiable and meets some standard assumptions (see, e.g., Deuflhard [34]), a root may be efficiently located using the classical Newton algorithm for a suitable initial guess $u^{(0)} := u_0 \in \mathbb{R}^n$. To this end, one iterates Newton's scheme for $n = 0$ until convergence:

$$\begin{aligned} d_u \mathcal{R}(u)|_{u^{(n)}} \left(\Delta u^{(n)} \right) &= - \mathcal{R}(u^{(n)}) && \text{(Linear solve)} \\ u^{(n+1)} &= u^{(n)} + \Delta u^{(n)} && \text{(Update)} \end{aligned}$$

Convergence only occurs for a suitable initial guess $u^{(0)}$ near a simple root $u^* \in \mathbb{R}^n$. In that case the convergence rate is quadratic. The quadratic convergence is clearly a local property, i.e., it depends heavily on a good choice for $u^{(0)}$. Thus, globalized Newton's methods have been invented in order to recover convergence in situations where the local Newton's scheme diverges; see Deuflhard [34] and references therein.

One possible globalization strategy is the family of homotopy methods for Newton's method. The idea is to introduce a sequence of starting values for which the Newton algorithm converges quadratically to useful intermediate solutions. To this end, one or multiple additional parameters are introduced. Let us focus on the case of a single homotopy parameter, say $t \in [0, T]$. This leads to a family $\mathcal{R} : \mathbb{R}^n \times [0, T] \to \mathbb{R}^n$ which we shall denote by $\mathcal{R}_t(u) := \mathcal{R}(u, t)$. Moreover, it is required that the original residual is recovered for $t = T$:

$$\mathcal{R}_T(u) = \mathcal{R}(u, T) = \mathcal{R}(u) \, .$$

Thus a root u_T^* of

$$\mathcal{R}_T(u_T^*) \; = \; 0 \; = \; \mathcal{R}(u_T^*)$$

also solves the original problem. Further, $\mathcal{R}_0$ corresponds to a nonlinear or linear problem which is simple (or at least possible) to solve.

Let us introduce a sequence of intermediate solutions $u_i^* := u_{t_i}^*$ corresponding to a strictly monotone sequence

$$0 = t_0 < t_1 < \ldots < t_i < \ldots < t_N = T, \quad 0 \leq i \leq N \, .$$

We denote the parameter increments by $\Delta t_i := t_{i+1} - t_i > 0$, for $i \geq 0$. The Newton algorithm for the computation of the next solution $u^*_{i+1} = u^*_{t_{i+1}}$ is then started with the last converged solution, i.e.,

$$u^{(0)}_{i+1} = u^*_i \ .$$

For the particular case of our mechanical problem, this ensures that it always stays close to its equilibrium of forces. In our experience, it is necessary to keep the system close to the equilibrium of forces. The growth of the polyconvex models here considered is extreme and evaluations far from the equilibrium of forces produce numerical exceptions immediately.

The introduction of a pseudo-time t parametrizing the loading, similar to the time dependent case, leads to an intuitive homotopy method called load stepping. More explicitly, let us consider the discrete balance of forces in an elastic body as a residual which we write, slightly abusing notation, as

$$\mathcal{R}(u,t) = G^{\text{int}}(u; \mathcal{BC}(t)) - G^{\text{ext}}(u; \mathcal{BC}(t))$$

depending on the displacement u and (in part implicitly) on the boundary conditions $\mathcal{BC}(t)$ which are parametrized by a pseudo-time $t \in [0,T]$.[3] We set

$$\begin{aligned} \mathcal{BC}_{\text{full}} &:= \text{ “ fully applied boundary conditions ” } , \\ \mathcal{BC}_{\text{rest}} &:= \text{ “ boundary conditions at rest ” } , \end{aligned}$$

$$\mathcal{BC}(0) = \mathcal{BC}_{\text{rest}} \quad \text{and} \quad \mathcal{BC}(T) = \mathcal{BC}_{\text{full}} \ .$$

In load-stepping, parametrizations of the boundary conditions are used to generate discrete homotopy paths of solutions $t_i \mapsto u^*_i$ through incremental loading. The mapping $t \mapsto \mathcal{BC}(t)$ is also called a loading curve.

Although in general quite successful, this strategy may fail in certain cases. For instance, a continuous path of solutions $t \mapsto u^*_t$ may lead to bifurcation points, e.g., when the structure starts to buckle. In such cases, the parallel solver `mparfeap` usually just breaks down in the load-stepping procedure due to a nearly singular tangent stiffness matrix. More specifically, we observe stagnation, i.e., the adaptively computed load step size at the i-th successful solution is then reduced with every failed step, here indexed by k to yield

$$\lim_{k \to \infty} \Delta t_{i_k} \to 0 \ .$$

Note that an optimal choice for Δt_i is not readily available, even if no singular points are encountered. One of the strengths of `FEAP` is that it allows the user to interactively explore the sensitivity of the simulation, e.g., for bifurcation analysis. It is generelly well-designed for exploration of nonlinear dynamics. In the current version of `mparfeap` is not capable to continue in such cases. Automated techniques to control the global nonlinear dynamics for large-scale problems in a more stable way are known, see, e.g., Wriggers [137], but they are outside the scope of this work. Let us just note, that such methods are not straight-forward to apply in the large scale.

1.3 Domain Decomposition Methods[4]

This section provides a highly condensed introduction to the FETI-DP method (**D**ual **P**rimal **F**inite **E**lement **T**earing and **I**nterconnecting), a non-overlapping domain decomposition method, used as a linear solution algorithm in a Newton-Krylov scheme.

[3] Note that Dirichlet boundary conditions are enforced by $\mathcal{A}_t := \mathcal{A}(\mathcal{BC}(t))$, i.e., the space of admissible configurations also depends on the pseudo-time $t \in [0,T]$.

[4] Parts of this section are based on a text prepared for a joint D-A-CH research proposal.

Domain decomposition methods constitute a thriving family of iterative solution methods for linear (or nonlinear) boundary value problems based on a geometric decomposition of the problem domain Ω into smaller subdomains Ω_i, $1 \leq i \leq N$. By restriction of the global solution function space V to the subdomains Ω_i one obtains induced local solution function spaces $V^{(i)}$ defined on the subdomains. This allows to formulate solution algorithms for boundary value problems in terms of the decoupled local solution spaces and certain coupling conditions linking the local solutions in such a way that a global solution can still be recovered.

Historically, the development of linear geometric domain decomposition methods can be traced back to the late 19th century, when Hermann A. Schwarz invented a domain decomposition method for the construction of explicit solutions for the Dirichlet problem [119]. His original method is well-known today as the alternating overlapping Schwarz method. However, due to the success of the direct methods in the calculus of variations the general interest in applying domain decomposition methods as an analytical tool diminished. The interest in domain decomposition methods then rekindled again in the context of finite element approximations of linear boundary value problems due to some particularly favorable properties, e.g.:

- the applicability of domain decomposition methods to finite element approximations of boundary value problems formulated on geometrically complex computational domains and in arbitrary dimensions,
- the robustness which is often observed for problems in structural mechanics, and
- the fact that domain decomposition methods lend themselves naturally to the construction of highly scalable parallel solution methods.[5]

An in-depth introduction to the broad subject of domain decomposition is not our goal here. Elegant and general introductory expositions of domain decomposition methods for partial differential equations are readily available in the monographs by Quarteroni and Valli [108], Smith, Bjørstad, and Gropp [124] and Toselli and Widlund [132]. There, detailed and concise account is given to the fundamental original contributions of the field and a wealth of useful references is provided.

It seems appropriate to mention at least some essential contributions and selected references for the following important families of domain decomposition methods:

- Overlapping Schwarz methods [119] (further, e.g., [132]),
- FETI-1 [44] (further, e.g., [17, 46, 47, 74, 94]),
- FETI-DP [49] (further, e.g., [48, 69, 71, 73, 75, 77, 79, 80, 95, 107]),
- BDDC [35] (further, e.g., [36, 83, 89, 90, 93, 96, 134, 135]; see also [30, 53]), and
- GDSW [39] (further, e.g., [38, 40, 41]).

Our focus here is only on non-overlapping domain decomposition methods of the FETI-DP type.

[5] This inspired the development of the parallel PETSc software library.

Let us first introduce some fundamental vocabulary:

Definition 1.3.1 (Non-Overlapping Domain Decomposition)**.** *A family of domains $\Omega_i \subset \mathbb{R}^n$, $i = 1, \ldots, N$ is called a* ***non-overlapping decomposition*** *of a domain $\Omega \subset \mathbb{R}^n$ into subdomains, if*

$$\Omega \;=\; \bigcup_{i=1}^{N} \Omega_i \quad \textit{and} \quad \mathrm{vol}(\Omega_i \cap \Omega_j) = 0 \;, \quad \forall\, 1 \leq i \neq j \leq N \;,$$

where vol *denotes the Lebesgue-measure in $\mathbb{R}^n$.*

A non-overlapping domain decomposition induces an internal interface separating the interiors of the subdomains Ω_i, $1 \leq i \leq N$.

Definition 1.3.2 (The Internal Interface)**.** *The* ***internal interface*** *associated to a non-overlapping decomposition of a domain Ω is given by*

$$\Gamma := \bigcup_{i=1}^{N} \partial\Omega_i \setminus \partial\Omega \;.$$

The decomposition of a computational domain Ω into a family of computational subdomains Ω_i, $i = 1, \ldots, N$ induces an embedding of the original global solution space V defined on Ω into a cartesian product of local spaces $V^{(i)}$ defined on Ω_i:

$$V \quad \hookrightarrow \quad \prod_{i=1}^{N} V^{(i)} \;\cong\; \bigoplus_{i=1}^{N} V^{(i)} \;. \tag{1.3.34}$$

There are different ways to partition the computational domain. In the present work, we have chosen to decompose cuboidal domains into cuboidal subdomains. This is a standard approach, commonly referred to as a structured domain decomposition. This eases the setup for weak scalability testing. For more complex geometries, e.g., arterial wall geometries, the sequential graph partitioner `METIS`, see Karypis, Schloegel, and Kumar [67], was used for the generation of the domain decompositions.

Consider now a computational domain Ω decomposed into $N \in \mathbb{N}$ nonoverlapping subdomains Ω_i, $1 \leq i \leq N$. FETI, and also any other geometric domain decomposition method, recovers a global solution $u \in V$ for a boundary value problem from local solutions $u^{(i)} \in V^{(i)}$ defined on the subdomains Ω_i. The interconnection of local solution patches along the internal interface Γ requires that the local solutions $u^{(i)}$ satisfy additional transmission conditions. In our scenario, we shall work with the Sobolev Hilbert spaces $V = H^1(\Omega)^3$ and $V^{(i)} = H^1(\Omega_i)^3$. A first natural transmission condition between local solution functions $u^{(i)}$ is continuity of u on the internal interface Γ. Under reasonable and standard assumptions this implies H^1-regularity for the global function u. To also satisfy the partial differential equation, the local solutions $u^{(i)}$ usually have to satisfy additional conditions on the interface. For example, if we consider a Poisson model problem and a decomposition into two subdomains, see Toselli and Widlund [132, Section 1.1], the normal derivatives of the local solutions must also be continuous.

We restrict our attention to finite element discretizations defined on the subdomains Ω_i which match on the subdomain interfaces. The local stiffness matrices $K^{(i)}$ and local load vectors $f^{(i)}$, $i = 1, \ldots, N$ can be *independently* assembled in parallel. Let us introduce the following notation

$$
\begin{aligned}
K &:= \bigoplus_{i=1}^{N} K^{(i)} = \begin{bmatrix} K^{(1)} & & \\ & \ddots & \\ & & K^{(N)} \end{bmatrix}, \\
u &:= \bigoplus_{i=1}^{N} u^{(i)} = \begin{bmatrix} u^{(1)} \\ \vdots \\ u^{(N)} \end{bmatrix}, \\
f &:= \bigoplus_{i=1}^{N} f^{(i)} = \begin{bmatrix} f^{(1)} \\ \vdots \\ f^{(N)} \end{bmatrix}.
\end{aligned} \tag{1.3.35}
$$

Note that the global degrees of freedom spanning V which are located on the interface Γ are split into multiple local degrees of freedom in different $V^{(i)}$. Accordingly, the discrete space for the iteration $\bigoplus_{i=1}^{N} V^{(i)}$ allows for multiple, arbitrary values of the local degrees of freedom located on the interface Γ. Thus, the iteration space $\bigoplus_{i=1}^{N} V^{(i)}$ contains finite element functions which are discontinuous on Γ. Only on convergence of the GMRES (or CG) algorithm the discrete solution u is single-valued and satisfies the transmission conditions on Γ. The degrees of freedom can be partitioned into interface variables Γ and interior variables I. This induces a Schur complement on every subdomain given by

$$
S^{(i)} := K_{\Gamma\Gamma}^{(i)} - K_{\Gamma I}^{(i)} \left(K_{II}^{(i)} \right)^{-1} K_{I\Gamma}^{(i)} \tag{1.3.36}
$$

and we can form the direct sum to obtain

$$
S := \bigoplus_{i=1}^{N} S^{(i)} = \begin{bmatrix} S^{(1)} & & \\ & \ddots & \\ & & S^{(N)} \end{bmatrix}. \tag{1.3.37}
$$

In the following two subsections, we introduce the FETI-1 and FETI-DP methods.

1.3.1 The FETI-1 Method

We begin by introducing the FETI jump operator $B = [B^{(1)}, \ldots, B^{(N)}]$ which measures how the local parts $u^{(i)}$, $i = 1, \ldots, N$, of $u = [{u^{(1)}}^T, \ldots, {u^{(N)}}^T]^T$ jump over the internal interface Γ. The entries of B are in $\{0, 1, -1\}$ and the operator computes differences of local values of u at matching finite element nodes on the interface Γ. Thus, the linear constraint

$$
Bu = 0
$$

enforces continuity of the displacement u on the interface Γ. It is instructive to consider first the case of a well-posed linear self-adjoint elliptic boundary value problem discretized by finite elements. This is the most extensively studied and accessible scenario. After finite element discretization, this problem can be cast into an unconstrained saddle point problem by introducing Lagrange multipliers λ as follows

$$
I(u, \lambda) := \frac{1}{2} u^T K u - f^T u + \lambda^T B u \,. \tag{1.3.38}
$$

Under the previously made assumptions K is a symmetric positive semi-definite matrix which can be shown to be positive definite on the kernel of B. Determination of the critical points leads to a linear system in saddle point form in the variables (u, λ) which reads:

$$
d_{(u,\lambda)} I(u, \lambda) = 0 \iff \left.\begin{array}{rcl} Ku + B^T\lambda &=& f \\ Bu &=& 0 \end{array}\right\}. \tag{1.3.39}
$$

This system can be solved by first eliminating the displacement variables u and solving the resulting Schur complement system by conjugate gradients. Note again that the block diagonal matrix K is in general only positive semidefinite. The general form for the solution can be written using a pseudoinverse K^+ of the stiffness matrix K and adding a suitable kernel element as follows

$$u = K^+(f - B^T\lambda) + R\alpha \, . \tag{1.3.40}$$

The column space of R spans the kernel of K, i.e., we have range$(R) = \ker(K)$ and α parametrizes the contribution lying in the null space of K. Note that the use of a pseudo-inverse is a typical feature of the FETI-1 method.

The next construction step in the FETI-1 method introduced in Farhat [43], Farhat and Roux [45] and Farhat and Roux [44] is then to enforce consistency of the linear system by use of a suitable projection. This allows a complete solution strategy based on a pseudoinverse K^+ of K. For a choice of symmetric positive definite matrix Q, e.g., $Q = I$, the projection is constructed as follows:

$$P_{\mathrm{FETI}-1}^T := I - G(G^TQG)^{-1}G^TQ, \quad G := BR \, .$$

This construction enforces orthogonality to the kernel of K which is an essential feature of the construction. Recall that the column space of R spans the kernel of K, i.e., we have range$(R) =$ $\ker(K)$. This can be considered as a drawback of the FETI-1 method, if the kernel of K is not explicitly known as it *can* be difficult to compute it numerically in a robust and reliable way.

The projection $P_{\mathrm{FETI}-1}$ provides FETI-1 with a coarse problem if the $K^{(i)}$ have a nontrivial kernel. In general, the matrix Q contributing to $P_{\mathrm{FETI}-1}$ is chosen such that the method is robust with respect to the choice of parameters, i.e., it has to accomodate coefficient jumps. In the FETI-1 method, the following equation has to be solved

$$P_{\mathrm{FETI}-1}^T \, \mathfrak{F}_{\mathrm{FETI}-1} \, \lambda = P_{\mathrm{FETI}-1}^T \, d \, , \tag{1.3.41}$$

with $\mathfrak{F}_{\mathrm{FETI}-1} := BK^+B^T$.

Since we do not use the FETI-1 method in the present text, we refer the interested reader to [132] for more details.

1.3.2 The FETI-DP Method

The FETI-DP (**D**ual **P**rimal **F**inite **E**lement **T**earing and **I**nterconnecting) method was first introduced in Farhat, Lesoinne, Le Tallec, Pierson, and Rixen [49]. It partitions the interface degrees of freedom on Γ into two disjoint sets: *primal* degrees of freedom and *dual* degrees of freedom.

By construction the primal variables $\tilde{u}_\Pi$ *exactly* satisfy a prescribed set of primal continuity constraints. This can, e.g., be achieved by a partial assembly operation which we denote by tilde operators. For example, $\tilde{u}$ denotes a vector satisfying the primal constraints *exactly* throughout iterations. On the dual interface degrees of freedom, the continuity constraint $Bu = 0$ is enforced through Lagrange multipliers λ, as in the FETI-1 method. The primal subspace provides FETI-DP with a coarse problem. The possibility of different choices for the primal coarse space entail a whole family of FETI-DP algorithms. Good scalability results in three dimensions in general require adequate coarse spaces. Let us introduce some possible strategies to select primal constraints. One may, e.g., impose vertex constraints, continuity constraints for edge and face averages, or continuity of higher order moments; see, for instance, Farhat, Lesoinne, and Pierson [48], Klawonn, Widlund, and Dryja [79], and Klawonn and Widlund [75, 76, 77].

We have used O. Rheinbach's implementation of FETI-DP presented in Klawonn and Rheinbach [70] and Rheinbach [110] as a nested linear solver component in a Newton-scheme. A so-called

transformation of basis approach allows to include the primal constraints on edge averages required by Algorithm C and Algorithm D_E which we shall introduce shortly; see, e.g., Klawonn, Widlund, and Dryja [79] and Klawonn and Rheinbach [70]. A detailed description of the approach and its implementation is available in the dissertation of Rheinbach [110]. In [72] the robustness properties of the transformation of basis approach are studied. For the historical development of the transformation of basis approach we refer to the introduction of Klawonn, Widlund, and Dryja [79], Klawonn and Widlund [77] and to Rheinbach [110].

Similar to the FETI-1 method, the FETI-DP method can be written as a saddle point problem[6]

$$\begin{bmatrix} \widetilde{K} & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \tilde{u} \\ \lambda \end{bmatrix} = \begin{bmatrix} \tilde{f} \\ 0 \end{bmatrix}.$$

Reflecting the restriction to a subspace satisfying the continuity constraint exactly in the primal variables u_Π, the matrix $\widetilde{K}$ and right hand side $\tilde{f}$ are partially assembled in the primal variables:

$$\widetilde{K} = \begin{bmatrix} K_{BB}^{(1)} & & & \widetilde{K}_{\Pi B}^{(1)T} \\ & \ddots & & \vdots \\ & & K_{BB}^{(N)} & \widetilde{K}_{\Pi B}^{(N)T} \\ \widetilde{K}_{\Pi B}^{(1)} & \cdots & \widetilde{K}_{\Pi B}^{(N)} & \widetilde{K}_{\Pi\Pi} \end{bmatrix}, \qquad \tilde{f} = \begin{bmatrix} f_B^{(1)} \\ \vdots \\ f_B^{(N)} \\ \tilde{f}_\Pi \end{bmatrix}.$$

Let us assume again, that K derives from a self-adjoint and elliptic bilinear form. In contrast to the previously presented FETI-1 approach, the local problems $K_{BB}^{(i)}$, $i = 1, \ldots, N$ can then be shown to be invertible given a suitable choice of primal variables, i.e., coarse problem. Thus, the primal variables in FETI-DP can be eliminated without using a pseudoinverse. The coupling in the primal variables provides a coarse problem for the method. Moreover, the favorable block structure of $\widetilde{K}$ still allows for a completely independent parallel factorizaton of the local problems $K_{BB}^{(i)}$, $i = 1, \ldots, N$. In particular, the system of equations can be reduced *in parallel* to an equation in λ and it finally remains to solve

$$\underbrace{B\widetilde{K}^{-1}B^T}_{=:\mathfrak{F}}\lambda = d. \tag{1.3.42}$$

This system is compactly written as

The FETI-DP Master System

$$\mathfrak{F}\lambda \;=\; d\,. \tag{1.3.43}$$

It is commonly referred to as the FETI-DP master system. Clearly FETI-DP is a dual method in the sense that one solves for the Lagrange multipliers λ. We may also write $\mathfrak{F} = B_\Gamma \widetilde{\mathcal{S}}^{-1} B_\Gamma^T$ where $\widetilde{\mathcal{S}}$ is obtained from $\widetilde{K}$ by eliminating all variables interior to the subdomains. B_Γ is obtained from B by discarding all columns corresponding to interior variables. This motivates the definition of the so-called Dirichlet preconditioner for $\mathfrak{F}$ given by

The FETI-DP Dirichlet Preconditioner

$$\mathfrak{M}_D^{-1} := B_{D,\Gamma} \widetilde{\mathcal{S}} B_{D,\Gamma}^T. \tag{1.3.44}$$

The matrices B_D and $B_{D,\Gamma}$ are scaled variants of the jump operator B. In the simplest case, the contribution from and to each interface degree of freedom is scaled by the inverse of the multiplicity

[6] Possibly *after carrying out a transformation of basis* which maps more general linear constraints, such as continuity of edge or face averages, to nodal constraints.

of the corresponding node. This is called multiplicity scaling. The multiplicity of a node is the number of subdomains it is contained in. For strongly heterogeneous problems more elaborate scalings defined by the coefficients ρ_i, are necessary; see, e.g., [77, p. 1532,Formula (4.3)] and [72, p. 1403, Formula (6)]. For our numerical experiments, only multiplicity scaling was considered.

For some model problems, including linear elasticity, the condition number of the FETI-DP method with the Dirichlet preconditioner $\mathfrak{M}_D$ can be explicitly estimated. Typically, a bound of the form

FETI-DP Condition Number Estimates

$$\kappa(\mathfrak{M}_D^{-1} \cdot \mathfrak{F}) = \kappa(B_{D,\Gamma}\widetilde{\mathcal{S}}B_{D,\Gamma}^T \cdot B_\Gamma \widetilde{\mathcal{S}}^{-1} B_\Gamma^T) \leq C\,(1 + \log(H/h))^2 \tag{1.3.45}$$

is obtained. Here h denotes the typical finite element diameter and H denotes the typical diameter of the subdomains Ω_i, $i = 1, \ldots, N$. The constant C is independent of h, H, and coefficient jumps.

For the case of a scalar second-order elliptic partial differential equation in two space dimensions, without coefficient jumps across the interface, a polylogarithmic bound has been shown in early work by Mandel and Tezaur [95]. The condition number estimate for the three-dimensional case and scalar second-order elliptic equations with discontinuous coefficients can be found in Klawonn, Widlund, and Dryja [79]. Using only vertex constraints as primal variables leads to a weaker bound for dual-primal FETI methods. This theoretical result can also be observed numerically; see Klawonn, Rheinbach, and Widlund [81] and Farhat, Lesoinne, and Pierson [48]. Theoretical extensions for linear elasticity can be carried out along the lines of the theory given in Klawonn and Widlund [77]. If edge averages (see definition below) are chosen as primal variables either additionally or instead of the primal vertex constraints, a quadratic-logarithmic bound can be obtained. For a proof in the case of scalar second-order elliptic equations, see Klawonn, Widlund, and Dryja [79] and for the more elaborate case of linear elasticity Klawonn and Widlund [77]. Note that these bounds can be extended to irregular subdomains such as those obtained from mesh partitioners. In Klawonn, Rheinbach, and Widlund [84], e.g., condition number bounds for linear elasticity in two space dimensions were proved for subdomains of quite general shapes, so-called Jones domains. To this end, the effects due to the wrinkled subdomain interfaces which are often produced by graph partitioning approaches have been estimated; see also Dohrmann, Klawonn, and Widlund [39] for related work.

We have already mentioned the attractive scalability properties of domain decomposition methods of the FETI type. A short description of the term scalability is hence appropriate here. Solution algorithms for linear equation systems are numerically scalable, if the computational costs of the solution process up to a prescribed accuracy, e.g., atol $= 10^{-9}$, grow at most linearly in the number of unknowns. More specifically, for domain decomposition methods using CG as the Krylov method this is the case if the condition number of the linear system is independent of the number of subdomains. If, additionally, the condition number grows only weakly, e.g., polylogarithmically, considered as a function of the local problem size H/h, we call the domain decomposition method **numerically scalable**.

Taking the general perspective of parallel computing, two other, slightly different notions of scalability, are commonly used. A parallel algorithm is **parallel scalable** if its runtime is inversely proportional to the number of processors used. For example, doubling the number of processor cores for a given fixed problem size ideally reduces the runtime by a factor of two. This property is also referred to as **strong scalability**. There is a second important notion. Suppose that the problem size and the number of processors are doubled simultaneously. If the runtime of a parallel algorithm is not affected, the algorithm is **weakly scalable**.

It is intuitive that numerical and parallel scalability are related and we consider the specific case of FETI methods.

Remark 1.3.3 (On Numerical and Parallel Scalability for FETI-DP methods)**.** *The typical poly-logarithmic bounds for the condition number of the preconditioned FETI-DP operator, e.g.,*

$$\kappa(\mathfrak{M}_D^{-1} \cdot \mathfrak{F}) \leq C\,(1 + \log(H/h))^2$$

imply numerical scalability. A bound of this type ensures that the condition number is bounded independently of the number of subdomains. Further, it is only weakly dependent on the number of degrees of freedom per subdomain which is measured by the quotient H/h. Numerical scalability is, however, just a necessary condition for parallel scalability. The reverse implication does not hold in general. Parallel scalability explicitly depends on the specific software implementation and the hardware infrastructure.

In the context of elliptic problems, both, the numerical scalability and the parallel scalability, can be shown to depend on a mechanism to transport information between the different subdomains during the solution process. This global coupling mechanism is represented by a so-called coarse space or coarse component. In FETI-DP methods this coarse space can, e.g., be induced by a choice of primal continuity constraints.

FETI-DP methods have been shown to be parallel scalable for up to 65 000 processor cores in Klawonn and Rheinbach [73]. For such parallel scalability, the coarse problem can in general only be solved inexactly, e.g., due to memory constraints. This approach is referred to as inexact FETI-DP; see, e.g., Klawonn and Rheinbach [71, 73]. In the context of this thesis, only exact FETI-DP algorithms are considered.

1.3.3 FETI-DP Coarse Spaces and Algorithms

In FETI-DP, the primal coarse space is induced by the partial assembly operation enforcing continuity constraints on the primal part of the interior interface Γ. For the computations in hyperelasticity which we present in Chapter 3, we experimented with three FETI-DP selection strategies for the primal degrees of freedom, i.e., with three different coarse spaces. To describe these in some more detail, we first introduce the sets of faces $\mathcal{F}$, edges $\mathcal{E}$ and vertices $\mathcal{V}$.

In the present work, we use a combinatorial definition of faces $\mathcal{F}$, edges $\mathcal{E}$ and vertices $\mathcal{V}$ as given, e.g., in Klawonn and Rheinbach [70, p.1528]. Towards a definition of the notions of faces, edges, and vertices induced by a domain decomposition Ω_i, $i = 1, \ldots, N$, we shall make use of a certain equivalence relation. Restricting ourselves to the nodes contained in $\partial\Omega$, $\partial\Omega_i$, $i = 1, \ldots, \mathbb{N}$ and Γ, we define

$$\mathcal{N}_x \;:=\; \{j \in \{1, \ldots, N\} : x \in \partial\Omega_j\} \tag{1.3.46}$$

for any interface nodal point $x \in \Gamma$. The set $\mathcal{N}_x$ collects the indices of all subdomains containing the node x. We call the number $|\mathcal{N}_x|$ the multiplicity of x. The associated nodal graph represents the nodal adjacency relations. For a node $x \in \Gamma$, we write $\mathcal{C}_{\text{con}}(x)$ for the connected component of the nodal subgraph induced by $\mathcal{N}_x$. For two interface points $x, y \in \Gamma$, we further define the following equivalence relation

$$x \sim y \quad \Longleftrightarrow \quad \mathcal{N}_x \;=\; \mathcal{N}_y \quad \text{and} \quad y \in \mathcal{C}_{\text{con}}(x). \tag{1.3.47}$$

Based on this construction, we can now introduce faces, edges, and vertices combinatorially as follows:

Definition 1.3.4 (Faces, Edges and Vertices)**.**

$$x \in \mathcal{F} \iff |\mathcal{N}_x| = 2\,. \tag{1.3.48}$$

$$x \in \mathcal{E} \iff |\mathcal{N}_x| \geq 3 \quad \textit{and} \quad \exists y \in \Gamma, y \neq x, \quad \textit{such that} \quad y \sim x\,. \tag{1.3.49}$$

$$x \in \mathcal{V} \iff |\mathcal{N}_x| \geq 3 \quad \textit{and} \quad \nexists y \in \Gamma, \quad \textit{such that} \quad x \sim y\,. \tag{1.3.50}$$

For the case of regular substructures, e.g., structured domain decompositions based on hexahedra or tetrahedra, this combinatorial definition of faces, edges, and vertices is consistent with our basic geometric intuition.

When approaching more realistic unstructured problems, the number of edges and potential edge constraints for some subdomains can sometimes be insufficient. In this case, one may, e.g., add constraints on some extra edges on $\partial\Omega$, which otherwise would be regarded as part of a face. A similar problem can appear for flat structures in which case additional constraints might be required for each subdomain. Let us summarize that the presented selection strategy for the edge nodes can be inadequate in certain cases. For a more in depth discussion including recovery strategies and alternative definitions, see Klawonn and Rheinbach [70].

We introduce the additional notations $\mathcal{F}^{ij}$, $\mathcal{E}^{ik}$, $\mathcal{V}^{il}$ to enumerate the faces, edges and vertices local to the subdomain Ω_i. This allows us to present some possible choices for the primal constraints, the coarse problem in FETI-DP, which are known to be scalable for linear elasticity in two or three dimensions. We follow the original presentation given in Klawonn, Widlund, and Dryja [79], where also the nomenclature Algorithm A, Algorithm B, ..., was coined.

The primal coarse space introduced in the foundational work on FETI-DP due to Farhat, Lesoinne, Le Tallec, Pierson, and Rixen [49] and analyzed in Mandel and Tezaur [95] is described by the following algorithm:

Definition 1.3.5 (Algorithm A [49])**.** *The primal coarse space is induced by enforcing continuity of u on all vertex degrees of freedom defined by $\mathcal{V}$ in Γ.*

It is well-known that this vertex-based Algorithm A scales well in two, but not in three space dimensions.[7] To improve scalability for three-dimensional problems, the coarse space of Algorithm A for FETI-DP is to be adapted or enriched, e.g., by adding edge averages, as in:

Definition 1.3.6 (Algorithm C [79])**.** *The primal coarse space is induced by enforcing continuity of u on* all *the subdomain vertices $\mathcal{V}$ and the continuity of* all *edge averages $\overline{u}_{\mathcal{E}^{ik}}$ taken over all edges $\mathcal{E}$ in Γ.*

The next algorithm is a particular variant of the more general Algorithm D introduced in Klawonn, Widlund, and Dryja [79] for elliptic model problems. It was consequently analyzed in Klawonn and Widlund [77] for the case of three-dimensional linear elasticity. The specific variant and notation D_E has been introduced in Klawonn and Rheinbach [70]. There, also the parallel scalability for linear elasticity was illustrated using an implementation of a transformation of basis approach.

Definition 1.3.7 (Algorithm D_E [70])**.** *The primal coarse space is generated by continuity of $\overline{u}_{\mathcal{E}^{ik}}$, i.e., by the continuity of the averages over all edges $\mathcal{E}$ in Γ.*

[7] This also shows up in our applications of Newton-Krylov-FETI-DP to three-dimensional hyperelasticity. It shall suffice here to report that Algorithm A was useful as an intermediate step to test the code.

1.3.4 Newton-Krylov-FETI-DP for Hyperelasticity

For the solution of problems in nonlinear elasticity, we have chosen to apply FETI-DP to the linearizations arising in a Newton scheme, i.e.:

Definition 1.3.8 (Newton-Krylov-FETI-DP)**.** *A **Newton-Krylov-FETI-DP** solver is an inexact Newton-solver using the FETI-DP method inside a Krylov subspace method for the iterative solution of the linear systems which determine the Newton corrections Δu.*

An analysis on the influence of iterative solution strategies on a Newton scheme was given by Dembo, Eisenstat, and Steihaug [32]. The usage of the implementation of the `FETI-DP-solver` by O. Rheinbach leads to a Jacobian-free method, i.e., it relies on matrix vector multiplications instead of forming an explicit Jacobian, see, e.g., the survey article by Knoll and Keyes [87].

The equilibrium of forces in our applications on hyperelasticity is governed by equation (1.2.18). It corresponds to a nonlinear functional $G(u)$ which is first linearized and then approximated by finite elements. This leads to a symmetric, but posssibly indefinite, tangent stiffness matrix $K(u)$ for the case of dead loads. For the case of follower loads, we have resorted to a Newton-like method by approximating the tangent stiffness matrix $K(u)$ with a matrix $K^S(u)$ obtained by reflecting the upper right triangle of $K(u)$ at the main diagonal. A similar strategy was previously applied by Pierson [107] to obtain a symmetric system matrix. In particular, this allows our FETI-DP implementation to profit from symmetric matrix storage in `PETSc` and on an LDL^T-type decomposition in the direct solver phases.

Remark 1.3.9 (FETI-DP for Non-Symmetric Indefinite Problems)**.** *Application of Newton's method to hyperelasticity requires the solution of linear boundary value problems in order to determine the Newton corrections Δu. In the presence of follower loads, the discretization of the equilibrium of forces leads, in general, to an unsymmetric and indefinite tangent stiffness matrix $K(u)$, see, e.g., [137] or our discussion on follower loads in Appendix A. To the best of our knowledge, there are no established condition number bounds for our scenario; see the chapter on indefinite and non-symmetric problems in Toselli and Widlund [132, Ch. 11] and references therein. For other work on the symmetric indefinite case in elasticity, see, e.g., Farhat, Li, and Avery [50] and Li and Tu [88] for BDDC. Note that the use of GMRES is in general necessary in the indefinite case.*

We have exclusively used GMRES without restart for our applications.

Polylogarithmic condition number estimates for FETI-DP, as previously presented, have been obtained for model problems including, in particular, linear elasticity in three-dimensions. Linear elasticity corresponds to a linearization of finite hyperelasticity in the reference configuration and thus to $K(u = 0)$. Experimental results as presented in Chapter 3 for $u \neq 0$ are thus of interest to shed more light on the kind of convergence number bounds that can be expected for finite strains. The `MPI`-parallel solver `mparfeap` presented in Chapter 2 is a Newton-Krylov-FETI-DP solver based on a slightly modified version of the `FETI-DP-solver` by O. Rheinbach for the computation of the Newton corrections. A simple but important improvement is the support of symmetric matrix storage and LDL^T decomposition. This reduces the memory requirements for the coarse problem, a key to obtain good scalability results which we present in Chapter 3.

As the present thesis is part of a body of work arising from an ongoing research collaboration, we want to mention some related work and shortly put it into perspective. The hyperelastic modelling presented in Brands, Klawonn, Rheinbach, and Schröder [23] enforces the incompressibility constraint of soft biological tissues approximatively by a penalty term. We follow this approach

here. Augmented Lagrangian algorithms have been explored in Brinkhues, Klawonn, Rheinbach, and Schröder [25] and in the dissertation by Brinkhues [24]. This approach allows the incompressibility constraint to be satisfied up to a prescribed numerical tolerance, which is of interest but outside of the scope of the present work. The $\overline{F}$-approach used in Brands, Klawonn, Rheinbach, and Schröder [23], Brinkhues, Klawonn, Rheinbach, and Schröder [25], Klawonn and Rheinbach [73] which is due to Simo [122] reduces volume locking effects. Numerically scalable FETI-DP algorithms for incompressible and almost incompressible *linear* elasticity have been proposed, but are not yet applied here. For an overview and current work on efficient coarse spaces for incompressible and almost incompressible linear elasticity, see, e.g, Gippert [55], Gippert, Klawonn, and Rheinbach [57] and Gippert, Klawonn, and Rheinbach [56].

A still recent development are nonlinear domain-decomposition based preconditioners acting on the level of the Newton scheme. Such approaches consequently lead to discontinuous Newton corrections. Nonlinear FETI-DP and BDDC algorithms have been investigated, e.g., by Klawonn, Lanser, and Rheinbach [86]; see also Klawonn, Lanser, Radtke, and Rheinbach [85]. Reduced communication costs are expected to yield algorithms scaling up to the exascale. For related work on applications of nonlinear domain decomposition methods in structural mechanics, see Gosselet and Rey [58, 59].

Another highly interesting modern approach to preconditioning is to adaptively compute optimal coarse spaces. This allows to improve the condition number of the preconditioned FETI-DP system in situations where not much is known about the operator a priori. Large reductions in the total number of FETI-DP iterations are possible. For recent work on adaptively computed coarse spaces in FETI methods, see Klawonn, Radtke, and Rheinbach [78] and references therein; see also Klawonn, Lanser, Radtke, and Rheinbach [85].

1.4 Tuning Newton-Krylov-FETI-DP in Load Stepping Strategies

In this section, we revisit load stepping as a homotopy method but in some more detail and with a view on our implementation in `mparfeap` presented in Chapter 2 and the applications in biomechanics presented in Chapter 3. In particular, we want to introduce two simple tuning options for Newton-Krylov-FETI-DP profiting from load stepping: linear extrapolation for the displacement u, and the choice of $\lambda^{(n-1)}$ as a dual initial guess for FETI-DP.

In applications in the continuum mechanics of soft biological tissues the arising root finding problems $\mathcal{R}(u) = 0$ are delicate. We think that it is not bold to say that the presented simulations of von Mises equivalent stress distributions in different layers of arterial wall structures in §3.2 with boundary conditions in the physiological regime are *hopeless* to be solved without an effective globalization method. In the present work, we have chosen to implement a homotopy method for the boundary conditions, i.e., load stepping, as the basic globalization method in `mparfeap`. This is, de facto, a standard approach in the engineering community.

To clarify the presentation, we first introduce two simple conventions:

Definition 1.4.1 (Loading Attempt)**.** *A launch of the Newton solver with a given load is called a* ***loading attempt****.*

Definition 1.4.2 (Load Step)**.** *A* ***load step*** *is a successful loading attempt, i.e., a loading attempt for which the implementation of the Newton solver signaled convergence.*

In particular, the sequence of load steps is obtained from the sequence of all loading attempts by simply removing all failed loading attempts. Further, we shall denote the index of:

- the loading attempt by a subscript $l \in \mathbb{N}$,

- the Newton step (in loading attempt l) by a superscript (n), $n \in \mathbb{N}_0$, and
- the GMRES iteration (in Newton step n) by a superscript (n,m), $n, m \in \mathbb{N}_0$.

To signify a converged quantity of an iterate scheme, we simply drop the corresponding iteration index, e.g., $\lim_{m\to\infty} \lambda_l^{(n,m)} = \lambda_l^{(n)}$ corresponds to the convergence of the GMRES for the preconditioned FETI-DP method.

Two interdependent aspects which determine the quality of a load stepping strategy are certainly:

- Robustness – i.e., the completion of the simulation with full loading.
- Performance – i.e., the total runtime which depends for the most part on:
  - Reductions and increments of the load step size.
  - The initial value for the Newton method.
  - The initial value for the Krylov method, i.e., for FETI-DP.
  - Fast detection and early abortion of scenarios with a high probability of wasted effort due to excessively "hard" nonlinear Newton or linear FETI-DP problems resulting from an overly optimistic loading increment Δt_l, $1 \leq l \leq L$.

Regarding the last point, we have made a few heuristic observations in our applications which we have tried to exploit, at least to some degree, in our adaptive load stepping strategy.

Remark 1.4.3 (Reduction of the Loading Increment and FETI-DP/GMRES)**.** *Let us consider the global stiffness matrix*

$$K_{(t_l+\Delta t_l)}(u_{l+1}^{(n)}) \,, \tag{1.4.51}$$

assembled for a given loading attempt at pseudo-time $t_l + \Delta t_l$ in Newton step n. The computational cost of a Newton step is determined by the cost for the solution of the associated preconditioned FETI-DP system. An intuitive measure for this is given by the number of GMRES iterations required to achieve convergence. Note that in our applications in biomechanics $m \geq 1000$ GMRES iterations were considered as "too expensive". The time complexity of a single GMRES orthogonalization step is quadratic in the number of Krylov vectors, hence iterations, i.e., in $O(m^2)$. Thus, speedups can *be achieved for a reduced loading increment if it leads to a sequence of global stiffness matrices*

$$K_{(t_l+\beta\Delta t_l)}(u_{l+1}^{(n)}) \,, \qquad 0 < \beta < 1 \,, \tag{1.4.52}$$

inducing a sequence of FETI-DP systems that are faster to solve. In general, getting closer to the last computed equilibrium of virtual work, i.e., choosing an initial value and boundary conditions closer to the last converged displacement u_l has exactly this effect. Heuristically, there are two main causes for a high number of GMRES steps: a large initial residual norm in comparison to the absolute tolerance for convergence, or a preconditioned linear FETI-DP system which is nearly singular. A small load reduction generally improves both scenarios. A decrease of the number of Newton steps is also to be expected. However, ***it can be very expensive*** *to waste a whole loading attempt due to a single expensive linear problem.*

Note that numerically singular systems are expected in the vicinity of bifurcation points. It is out of the scope of the present work to treat load stepping schemes that are robust in the presence of nonlinear instabilities such as buckling. Further, there is a possibility to overshoot on instabilities which selects a somewhat arbitrary bifurcation branch. This is currently not detected.

We consider two general strategies to compute load increments: first, the Δt_l can be prescribed a priori; second, they can be adaptively computed based on the progress of the simulation. In general, one strives for a homotopy sequence simultaneously maximizing robustness and minimizing the computational time. To this end, the pseudo-time increments Δt_l should be large, but only so

that the started Newton iterations are, ideally, still quadratically convergent. Two load stepping strategies are currently available in the parallel solver `mparfeap` presented in Chapter 2.

Load Stepping Strategies in `mparfeap`

1. Linear strategy:
   - User supplied number of load steps $N \in \mathbb{N}$.
   - $\Delta t_l = \frac{1}{N}, \quad 1 \leq l \leq N.$
2. Adaptive strategy:
   - *Expansion on success* of the Newton method:
   $$\Delta t_{l+1} \;=\; \alpha \;\cdot\; \Delta t_l, \qquad \alpha > 1\,. \tag{1.4.53}$$
   - *Reduction on failure* of the Newton method:
   $$\Delta t_{l+1} \;=\; \beta \;\cdot\; \Delta t_l, \qquad 0 < \beta < 1\,. \tag{1.4.54}$$
   - User supplied estimate of required load steps for linear strategy $N \in \mathbb{N}$:
   $$\Delta t_1 = \frac{1}{N}. \tag{1.4.55}$$
   - User supplied delay $d \in \mathbb{N}$: expansion of Δt_l after d load steps.

Note that a slightly different adaptive load stepping strategy was applied in Brinkhues, Klawonn, Rheinbach, and Schröder [25] and Brinkhues [24] which is, however, not currently available in `mparfeap`.

For the applications to realistic problems in the biomechanics of arterial walls presented in Chapter 3 *we have used the adaptive load stepping strategy.* The convergence criteria are standard, so let us focus on the failure criteria. There are multiple heuristic criteria which trigger a reduction of the load step size in `mparfeap`.

Adaptive Strategy: Failure Criteria for a Loading Attempt

- Maximal number of Newton iterations exceeded.
- Maximal number of FETI-DP iterations exceeded.
- Numerically singular tangent, i.e., the preconditioned FETI-DP operator satisfies
  $$\mathrm{cond}_2^{\mathrm{est}}\left(\mathfrak{M}_D^{-1}\mathfrak{F}\right) \quad > \quad 10^6\,.$$
  A condition number estimate is computed by the GMRES implementation in `PETSc`.
- Local orientation reversal or degeneration, i.e., the displacement $u_l^{(n)}$ corresponds to a deformation mapping φ satisfying
  $$\det[F] \leq 0, \quad F := \nabla\varphi$$
  in a cubature node of a finite element T.
- Limit for the initial residual in the Newton method exceeded, i.e.,
  $$||r_{\mathrm{Newton}}(u_l^{(0)})||_2 \quad > \quad 10^6$$

- Limit for the initial residual in the FETI-DP/GMRES method exceeded, i.e.,

$$||r_{\text{FETI-DP}}(\lambda_l^{(n,0)})||_2 \quad > \quad 10^6 \; .$$

- Failure to assemble the stiffness matrix or residual. For example, the exceptional IEEE754 double values $\pm\infty$ and `NaN` are detected in

$$r_{\text{Newton}}(u_l^{(0)}), \quad r_{\text{FETI-DP}}(\lambda_l^{(n)}) \quad \text{and} \quad K(u_l^{(n)}) \; .$$

There are two reasons to reject loading attempts which signal one of the previously listed events, either this increases the robustness or it prevents scenarios where a high number of Newton or FETI-DP iterations is necessary for convergence. The tradeoff are additional loading attempts, which *can* be very expensive. Let us shortly mention two implementation details of `mparfeap`. In general `mparfeap` aborts failed loading attempts as soon as they are detected. For example, the failure criterion for the estimated condition number aborts the loading attempt *before* the `PETSc` GMRES exceeds the maximal number of iterations, i.e., after the iteration for which it was detected. The check for the initial GMRES residual aborts the linear solver after this residual is available, i.e., immediately after the first GMRES iteration.

1.4.1 Linear Extrapolation of the Displacement

In our applications in biomechanics, multiple load steps are in general necessary, see Chapter 3. It is intuitive to take a subsequence of solutions computed in previous load steps u_{l_i}, $0 < l_1 < l_2 < \ldots < l$ to predict the next solution. This estimate $u_{l+1}^{(0)}$ can then be used as an educated initial guess for Newton's method in the next load step. A simple but effective possibility is certainly to linearly extrapolate the last two converged displacements u_{l-1} and u_l. This method which was kindly communicated to us by Schröder [113] proved to be extremely helpful for our weak scalability results in §3.4.

This seemingly well-known technique can be derived as follows. Suppose that Taylor's theorem is applicable to the quasi-static solution $u : [0, T] \to V$ considered as a function of the homotopy parameter t. Then, we may expand

$$u_{l+1} = u(t_{l+1}) = u(t_l + \Delta t_l) = u(t_l) + \Delta t_l \left.\frac{du(t)}{dt}\right|_{t=t_l} + O(\Delta t_l^2) \; . \tag{1.4.56}$$

Let us now estimate the derivative by a left sided difference quotient

$$\left.\frac{du(t)}{dt}\right|_{t=t_l} \approx \frac{(u_l - u_{l-1})}{\Delta t_{l-1}} \; . \tag{1.4.57}$$

This leads to a first order extrapolation formula given by

Linear Extrapolation for the Displacement

$$u_{l+1}^{(0)} \; := \; u_l + \Delta t_l \, \frac{(u_l - u_{l-1})}{\Delta t_{l-1}}, \qquad \text{with} \quad u_1^{(0)} := 0. \tag{1.4.58}$$

The so-called Euler-Newton method, a simple continuation method, is similar; see, e.g., the monograph by Allgower and Georg [3]. It is also possible to use quadratic or cubic extrapolation, but higher order extrapolation schemes are currently not implemented in `mparfeap`.

Although it can be very effective in elasticity, we want to stress that this linear extrapolation technique is *well-known to fail in some important scenarios*, most prominently for plasticity; also

communicated by Schröder [113]. This may be due to the requirement of differentiability of $u(t)$ w.r.t. the pseudo time t in the given derivation.

1.4.2 A Nonzero Initial Guess for λ in Newton-Krylov-FETI-DP

Similar to the choice of a good initial guess for the Newton scheme itself, it is also interesting to generate good initial guesses for the nested Krylov subspace solver in the individual Newton steps. A FETI-DP solver iterates on the variables enforcing the continuity constraint on the dual part of the interface Γ. It solves a FETI-DP master system for the Lagrange multipliers λ which is of the general form

$$\mathfrak{F}\lambda \;=\; d\,.$$

For a derivation of the FETI-DP master system including also the preconditioner, we refer the interested reader to §1.3 and further references therein.

We shall denote the index of the loading attempt by $l \in \mathbb{N}$, the index of the Newton step by $n \in \mathbb{N}_0$ and the index of the GMRES iteration by $m \in \mathbb{N}_0$. Introducing the dependency on u, we thus have to solve the FETI-DP master system

$$\mathfrak{F}|_{u_l^{(n)}}\,\lambda_l^{(n)} \;=\; d|_{u_l^{(n)}}\;. \tag{1.4.59}$$

Usually, the initial guess for the GMRES iteration is just set to zero $\lambda_l^{(n,0)} = 0$, for all Newton steps $n \in \mathbb{N}_0$, but let us try to improve on this.

Suppose that the global finite element tangent stiffness matrices $K(u)$ depend continously on $u \in B_\varepsilon(u_l) \subset V$, i.e., for u sufficiently close to the converged solution u_l. Then, upon convergence of the Newton scheme we have

$$\lim_{n\to\infty} u_l^{(n)} = u_l\;. \tag{1.4.60}$$

At the same time the tangent stiffness matrices converge as well

$$\lim_{n\to\infty} K(u_l^{(n)}) = K(\lim_{n\to\infty} u_l^{(n)}) = K(u_l)\;. \tag{1.4.61}$$

This is consistent with the fact that the Newton corrections converge to zero. Assuming furthermore the continuous invertibility of (1.4.59) as a function of $u \in B_\varepsilon(u_l)$ we conclude the convergence of the Lagrange multipliers, i.e.,

$$\lim_{n\to\infty} \lambda_l^{(n)} = \lambda_l\;. \tag{1.4.62}$$

Under these assumptions, for n large enough, convergence sets in and $\lambda_l^{(n-1)}$ and $\lambda_l^{(n)}$ must be close to each other. It is, then, quite intuitive to initialize the GMRES iteration for $\lambda_l^{(n)}$ in a given Newton step $n > 1$ with the linear solution $\lambda_l^{(n-1)}$ obtained in the last Newton step. This is exactly the strategy we propose here:

Nonzero Initial Guess for λ

$$\lambda_l^{(n,0)} \;:=\; \lambda_l^{(n-1)}\,, \qquad \text{with} \quad \lambda_l^{(0,0)} \;:=\; 0\,. \tag{1.4.63}$$

One may expect to see reductions in the initial FETI-DP GMRES residuals during the Newton iteration and a reduction of the number of GMRES iterations. This is indeed apparent in the convergence histories produced by our numerical experiments; see §3.5 where also quantitative gains are reported.

1.4.3 Some Remarks on Globalization Strategies for Newton Methods

It is a standard result in numerical analysis that Newton's method is *locally* quadratically convergent, i.e., for initial guesses near a simple root, see, e.g., Quarteroni, Sacco, and Saleri [109] for a precise statement. Globalized Newton methods restore or speed up convergence of the Newton scheme for bad initial guesses far from a root. They may also continue on bad intermediate iterates, e.g., singular points. The necessary additional ingredient is global information about the problem. Note that quadratic convergence is not to be expected for globalized Newton methods; see, e.g., Deuflhard [34] or Nocedal and Wright [105].

The present work is based on load stepping methods as a globalization strategy. These are homotopy methods for the boundary conditions. Another important family of globalization methods is obtained by appropriate damping or amplification of the Newton corrections Δu in case of bad starting values for the Newton scheme. This family of globalization strategies is commonly referred to as the family of *line search techniques*, see, e.g., Dennis Jr. and Schnabel [33] or Nocedal and Wright [105].

We want to make some remarks regarding structural properties of the root finding problems arising from the considered boundary value problems in mechanics which are of some relevance for the application of a nested line search in the Newton solver. For simplicity, let us neglect the implications of the $\overline{\mathrm{F}}$-approach which does affect the interpretation of the stiffness matrix as a derivative.

There are then two possible scenarios:

- Only Conservative Loadings: the tangent stiffness matrices $K(u)$ are, *possibly indefinite*, but symmetric Hessian matrices of a mechanical potential energy.
- Non-Conservative Loadings: there is no potential energy and the Hessian structure is entirely lost. This is essentially a root finding problem. One may consider $\frac{1}{2}||r_{\mathrm{Newton}}(u)||^2$ as a pseudo-energy, however, we see no immediate interpretation in mechanics.[8]

For the case of indefinite Hessians the Newton search direction is not always a descent direction for the energy. Many standard line search techniques, however, assume this to be the case. Further, assumptions on the monotonicity of the residual may be invalid. For example our computations of arterial wall stresses produce increasing residual norms during the initial phase of some Newton iterations. Thus, e.g., non-monotone Armijo strategies may be of interest, see Grippo, Lampariello, and Lucidi [60].

Further, for applications presented in Chapter 3 we have found that the growth of the discretized stored energy is often extreme to the extent that the machine arithmetic is exceeded. This *may* even be related to the growth conditions which are typical for polyconvex strain energies. This is not uncommon. For example Wriggers [137, p.258] report for problems in hyperelasticity that the quantity

$$\log(\det[K(u)]) \quad = \quad \pm\infty$$

in IEEE754 double representation. We have also found that the stiffness matrices can only be assembled for conservative choices of the starting values of the Newton scheme. In short there are hard limits to what can be done if the load step size is *too* optimistic and it can be impossible to recover from that using a line search technique.

The potential gains of optimal embedded line search techniques used in an adaptive load stepping strategy are still not entirely clear to us. For example, the line search parameters might be coupled with the adaptive load stepping in intricate ways. Our preliminary experiences with nested line search approaches were mixed. It turned out to be difficult to optimize robustness and performance

[8] Note that all line search strategies in FEAP are based on this residual norm, i.e., not on a mechanical potential.

of an adaptive load stepping method with a nested line search in the Newton solver. We shall not report on specific results here as they are preliminary. A second well-known family of globalization strategies which we have not yet considered are so-called trust region strategies, see e.g., Nocedal and Wright [105].

We summarize that we have based our numerical experiments on a failure-based adaptive load stepping strategy mainly to increase robustness, but with good performance in mind.

In the next chapter we continue with a description of a parallel scalable Newton-Krylov-FETI-DP software framework based on `FEAP` and FETI-DP.

CHAPTER 2

A Parallel Solver Framework Based on **FEAP**

The aim of this chapter is to present a multi-language MPI parallel software framework (F77/C/C++) consisting of multiple software components and some of the development methodologies which we have applied. This framework is to be considered as a major contribution of this thesis. The main objective of this new software infrastructure is to close a software gap between the two previously existing scientific software components

FEAP-JS [sequential]	and	**FETI-DP-solver** [parallel]

in a new MPI parallel and scalable Newton-Krylov-FETI-DP solver. By FEAP-JS we denote an extended version of FEAP which is actively developed by the Institute of Mechanics at Universität Duisburg-Essen lead by J. Schröder. It provides multiple extensions. In our context the most important addition are finite element-based implementations of anisotropic and incompressible material laws which are suitable for the description of the biomechanics of soft biological tissues. As part of his dissertation, O. Rheinbach developed a highly scalable PETSc-based MPI parallel FETI-DP implementation; see Rheinbach [110]. In the current thesis this software component will be referred to as the FETI-DP-solver.

These two components, once integrated, allow for fully MPI parallel simulations and to research the scalability properties of such simulations in applications to the biomechanics of soft tissues superseding a previously developed semi-parallel framework. Such a Newton-Krylov-FETI-DP solver has been developed and we shall introduce it here as mparfeap. For applications in biomechanics, see Chapter 3.

2.1 Component Overview

This short overview of the different scientific software components and their collaboration pattern is meant to set the stage for the subsequent detailed technical discussion. Furthermore, in passing, we introduce a glossary of relevant software components.

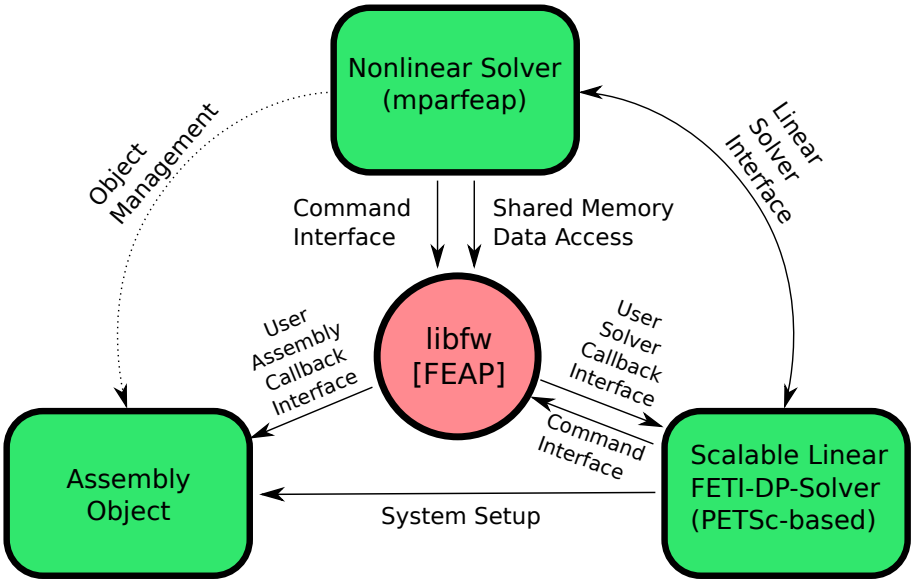


Figure 2.1: Schematic component collaboration diagram for a nonlinear Newton-Krylov-FETI-DP-solver based on FEAP. This example shows how the various interfaces provided by a new FEAP interface library libfw are used by mparfeap for fully MPI parallel assembly based on FEAP.

An overview of the developed parallel software framework is displayed in the collaboration diagram in Figure 2.1. It shows an encapsulated FEAP (or, more precisely, FEAP-JS) instance which communicates with the other components using direct and callback interfaces. The two callback interfaces are built on top of the standard user solver extension points of FEAP. More precisely, the user assembly callback interface is built on the user assembly interface extension point uasble_ and the user solver callback interface is built on the user assembly interface extension point uasble_.

The Cornerstones – PETSc and FEAP: The developed software framework makes extensive use of the following established large-scale scientific software projects:

Component	Version	Language	Initiators	Institution
PETSc	2.3.3-p{15,16}	C/C++/(F77)	Satish Balay, William D. Gropp, Lois Curfman McInnes, Barry F. Smith	Argonne National Lab.
FEAP	8.2	F77/C	R. L. Taylor et al.	UC Berkeley

FEAP is the **F**inite **E**lement Analysis **P**rogram: [129, 130]. The FEAP software package is described in the introduction of the official FEAP user manual [129] as follows:

> *"The Finite Element Analysis Program (FEAP) is a computer analysis system designed for:*
>
> 1. *Use in course instruction to illustrate performance of different types of elements and modeling methods;*
> 2. *In a research, and/or applications environment which requires frequent modifications to address new problem areas or analysis requirements."*

PETSc is the **P**ortable, **E**xtensible **T**oolkit for **S**cientific **C**omputing: Balay, Gropp, McInnes, and Smith [5], Balay, Adams, Brown, Brune, Buschelman, Eijkhout, Gropp, Kaushik, Knepley, McInnes, Rupp, Smith, and Zhang [6, 7].

A succinct description of PETSc written by the developers can be found on the PETSc webpage. We quote:

> *"PETSc, pronounced PET-see (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, shared memory pthreads, and NVIDIA GPUs, as well as hybrid MPI-shared memory pthreads or MPI-GPU parallelism."*

Previous Infrastructure – *Independently* Developed Components: In the research groups of A. Klawonn and J. Schröder the following two components were *independently* developed on the basis of the before-mentioned frameworks PETSc and FEAP. These components were to be coupled for combined simulations.

Software Component	Programming Language(s)	Short Description
FEAP-JS	F77/C	Extended version of FEAP providing, e.g., polyconvex models for soft biological tissues.
FETI-DP-solver	C/C++	Highly scalable MPI-implementation of FETI-DP based on PETSc.

Previous Infrastructure – *Jointly* Developed Components: As a first step, a semi-parallel software framework was developed in the context of DFG project KL 2094/1-1,2, SCHR 570/7-1,2 to test the algorithmic approaches and to pave the way for the envisaged massively parallel simulations. This semi-parallel framework was developed in a joint effort and consists of two refined components, which we shall denote as follows:

Software Component	Programming Language(s)	Short Description
FEAP-JS-FETI-DP	F77/(C)	An extended version of FEAP-JS which uses the FETI-DP-server for the parallel solution of the linear systems.
FETI-DP-server	C/C++	UNIX sockets based server application launching the parallel FETI-DP-solver.

Parallel Infrastructure – *Contributed* Components: For the realization of massively parallel simulations on a Cray XT6 the following new components were developed for KL 2094/1-{1,2}, SCHR 570/7-{1,2} and in prospect of a followup D-A-CH project on fluid-structure interaction based on LifeV and FEAP. These components form an integral part of the contributions of the current thesis:

Software Component	Programming Language(s)	Short Description
FEAP-FW	F77/C	An extended version of FEAP with interfaces for external control and data access.
libfw	C	FEAP wrapper library (uses FEAP-FW extensions).
fddp	C++	FEAP domain decomposition data processor command line application.
libfddp	C++	FEAP domain decomposition data processor utility library.
mparfeap	C++	Massively parallel FEAP (uses the FETI-DP-solver).

These components were developed by the present author. libfddp was developed by the present author and in parts by O. Rheinbach.

2.2 Component Integration in Stages: FEAP-JS and the FETI-DP-solver

The integration of the customized FEAP installation FEAP-JS with the highly scalable parallel FETI-DP-solver implemented in PETSc proceeded incrementally in multiple stages. Four major stages can be identified. The work here presented was begun after the two integration stages I/II were already completed; see Table 2.1.

Integration Stage	Data Interface	#FEAP Instances	#Solver Instances	#Processes (total)
I	Filesystem	1 (sequential)	N (parallel)	N + 1 (separated)
II	UNIX-Sockets	1 (sequential)	N (parallel)	N + 1 (separated)

Table 2.1: Previously realized integration stages for FEAP and the highly scalable FETI-DP-solver by O. Rheinbach. Note that FEAP and the FETI-DP-solver were started as separate UNIX-processes. The finite element stiffnesses were computed sequentially in a single instance of FEAP and the tangent systems were solved in N additional MPI processes.

In the semi-parallel integration stages I and II, the element stiffness matrices and right hand sides were copied across UNIX process boundaries from a FEAP master process to the PETSc solver slave processes which were started in parallel. The slave processes were started and controlled by a dedicated FETI-DP-server application. In every Newton iteration the linearized system was sequentially assembled in the FETI-DP-server and then solved in parallel using the FETI-DP-solver. After the linear solution phase, the FEAP master process received the Newton correction from the FETI-DP-server and initialized the next iteration.

Since the framework in integration stages I/II was only semi-parallel, the simulations in the transitional integration stages I and II were – by construction – not parallel scalable. In practice the sequential assembly phase in FEAP turned out to be a quite severe bottleneck. The simulations based on that framework were reasonably limited to a small cluster or workgroup server. A full parallelization of the assembly phase was clearly the next essential milestone towards large-scale parallel computations.

Remark 2.2.1 (Control Flow in Integration Stages I/II)**.** *In the previous semi-parallel framework FEAP acted as the master application, i.e., the simulations were entirely controlled by FEAP. The master FEAP process was initalized using FEAP input cards (in the usual way) and passed control to the FETI-DP-server during the assembly and linear solution phases. Special FEAP user macros (FETI) provided triggers to communicate with the parallel FETI-DP-server via UNIX sockets.*

Contribution – Parallel and unified component integration: For the final phase of the research project massively parallel computations of physiological arteries were scheduled. The requirements for the research project demanded a highly scalable software framework. This corresponds to the final integration stage IV:

Integration Stage	Data Interface	#FEAP Instances	#Solver Instances	#Processes (total)
III	Filesystem	N (parallel)	N (parallel)	N (unified)
IV	Shared-memory	N (parallel)	N (parallel)	N (unified)

Table 2.2: The parallel integration stages. The final integration stage IV corresponds to a fully MPI parallel scalable unified shared-memory integration of FEAP-JS with the highly parallely scalable FETI-DP-solver developed by Rheinbach [110].

Although a fully MPI parallel implementation, the transitional integration stage III was not scalable due to filesystem-based data sharing. Distributed filesystems and the associated storage

hardware cannot handle the massive amount of data generated by finite element analysis during the solution process in a scalable way, but the availability of all system matrices eased debugging of the integration considerably.

Remark 2.2.2 (Control Flow in Integration Stages III/IV)**.** *In contrast to integration stages I and II, the parallel integration stages III and IV necessitated an external access to the control flow of* FEAP*. This possibility to remote control* FEAP *is the most essential ingredient allowing for the flexible integration of* FEAP *as a third party software library into other software packages.*

2.3 FEAP: Its Usage, Implementation and Extensibility

Let us now present the software package FEAP in some more detail.

The Software Package: FEAP is an actively developed and well-established software package for finite element-based simulations. It is used extensively in linear and finite structural mechanics. The center of the development efforts is the research group of Richard L. Taylor at UC Berkeley, the original author of FEAP. This group manages the releases, distribution and support of the official versions of the FEAP software package. In the context of this thesis it is fundamental to note that the source code is part of the official FEAP distribution. Every licensed user of FEAP receives a copy of the source code and is free to add extensions. Due to its long development history, FEAP is certainly one of the most well-tested finite element simulation environments for structural mechanics applications.

Programming Languages (F77/C): For the largest part, FEAP is written in the F77 programming language. Some particular modules have been realized using the C programming language, as they need low-level access to the underlying operating system. Let us mention two important examples, which we shall revisit shortly: the memory management subsystem code and the X11 graphics rendering code.

Documentation: The FEAP software package is very powerful and feature-rich and the developers provide very detailed documentation on the usage of the program features in the FEAP User Manual [129] together with extensive documentation of demo examples in the FEAP Example Manual [126]. Internal implementation details and user extensions are discussed in the FEAP Programmers Manual [127]. The theoretical foundations of the implemented algorithms are described in the FEAP Theory Manual [128]. For our discussion, it thus suffices to focus on implementation aspects relevant to the software framework to be presented here and particular weight is given to internal aspects of FEAP which are not discussed in the officially distributed manuals.

Interaction Modes: Let us give a short introduction on how FEAP is used. The user interaction with the command line application FEAP is centered about a so-called FEAP *input card* which can also toggle the input stream to interactive input. First, during the startup of a FEAP session, FEAP parses such a user specified *input card*. This card is stored as a plain text ASCII-file on disk and consists of text blocks in FEAP command language syntax, see Figure 2.2 for an example. These blocks are also called records. FEAP can now work either in batch mode or in interactive mode. In batch mode the FEAP input is read exclusively from the FEAP input card used for startup. This is the default mode. In interactive mode[1] FEAP provides an interactive shell to the user. Using this shell the FEAP user can interactively control and analyse a running simulation. This is very interesting, for example, to carry out manual bifurcation analysis, a task which is quite hard to

[1] Interactive mode is activated by an INTEractive statement in the FEAP input card.

automate and directed user intervention may be far superior to any automated approach. In both modes, it is furthermore possible to execute user script files which are stored in plain text files with the file suffix `.pcd`.

Example for a FEAP input card

Listing 2.1:

```
FEAP * * 4-Element Patch Test
  9,4,1,2,2,4
MATErial,1
  SOLId
    PLANe STRAin
    ELAStic ISOTropic 1000.0 0.25
                    ! Blank termination record
COORdinates
  1 0  0.0  0.0
  2 0  4.0  0.0
  3 0 10.0  0.0
  4 0  0.0  4.5
  5 0  5.5  5.5
  6 0 10.0  5.0
  7 0  0.0 10.0
  8 0  4.2 10.0
  9 0 10.0 10.0
                    ! Blank termination record
ELEMents
  1 1 1 1 2 5 4
  2 1 1 2 3 6 5
  3 1 1 4 5 8 7
  4 1 1 5 6 9 8
                    ! Blank termination record
BOUNdary restraints
  1 0 1 1
  4 0 1 0
  7 0 1 0
                    ! Blank termination record
FORCes
  3 0 2.5 0.0
  6 0 5.0 0.0
  9 0 2.5 0.0
                    ! Blank termination record
END

BATCh
  FORM residual
  TANGent
  SOLVe
  DISPlacement ALL
  STREss ALL
  PLOT STRE 1
  PLOT REACtions
END

STOP
```

]

Figure 2.2: This example input file `Ex1/Ipatch_4el` is part of the official `FEAP` v8.2 examples package. A detailed explanation can be found in the official `FEAP` examples manual [126]. Note the general block-based structure.

Extensibility and Customized Installations: An inspection of the actual FEAP code or the FEAP programmer manual [127] makes it obvious that FEAP has been carefully designed from the ground up with user extensibility in mind. The user base of FEAP consists mostly of research groups in continuum mechanics, mostly in structural mechanics and closely related disciplines. Most research groups using FEAP add custom functionality to the official distribution of FEAP. Over the years a **customized user installation** of FEAP is created by common efforts. A customized user installation often extends the base functionality significantly, but typically leaves the core infrastructure in FEAP untouched. Its extensibility is certainly one of the main advantages of FEAP and contributes a lot to its popularity.

Extensibility: To ease extensibility FEAP provides some clearly specified and well-documented mechanisms and interfaces for user extensions. These extension points are documented in the FEAP programmer manual [127]. Note again that FEAP is distributed as source code and thus the user is not in a fundamental way limited to use the before-mentioned predefined extension mechanisms. For the implementation of the FEAP-wrapper library libfw, for example, we use the available user extension mechanisms as well as other modifications. The libfw library hooks into the so-called FEAP "user solver"-interface, which is typically used to add a third party direct or iterative linear solver to FEAP. This is accomplished by implementing the following two standard FEAP user extension interfaces:

FEAP user extension points used by libfw

- User assembly interface [uasble.f]
- User solver interface [usolve.f]

This extension mechanism, as documented in the official FEAP programmers manual [127], is activated only in "user solver mode". It allows for the external assembly of element contributions via uasble_ and to completely externalize the linear solver phases via usolve_. This mode is selected by setting the internal variable solver = .false. in the program startup code. This disables the FEAP-internal direct solver and in turn enables usage of the user solver.

After these introductory notes of some fundamental aspects of FEAP, we progress now to the requirements that our parallel software framework should satisfy. To this end, we illuminate the development process for the entire framework and describe in passing also the most important design decisions that were made.

2.4 Requirements Analysis for the Parallel Software Framework

The first step in the planning phase of any larger software project is a detailed and careful analysis of the requirements for the solution to be developed. In close collaboration with D. Brands (FEAP -JS) and O. Rheinbach (FETI-DP-solver) the following requirements for the planned parallel software framework were identified:

Feature requirements

- Integration of FEAP with the FETI-DP-solver in a massively MPI-parallel scenario.
- Flexible access to the control flow of FEAP.
- Shared-memory access to data stored in FEAP.
- Unified integration in one UNIX-process per MPI-instance.
- Support for customized FEAP user installations, for example, FEAP-JS.[a]
- Highly scalable and robust parallel assembly.
- Flexible assembly of multiple subdomains per MPI process.

[a] FEAP-JS provides implementations of polyconvex, anisotropic and incompressible models for soft-tissues.

Maintainability requirements

- Minimal changes to FEAP [v8.2].
- Support for future versions of FEAP and customized user installations, in particular for FEAP-JS.
- Detailed tracking and documentation of all modifications done to FEAP.

Usability requirements

- Support the currently established development and testing workflow of the FEAP-JS-developers.
- The FEAP component supports inclusion into third-party software, e.g., into LifeV.
- Focus on flexible command line tools and software libraries for all developed components.

Testing requirements

- Support for established testing workflow of FEAP-JS-developers.
- Minimal duplication of testing efforts for functionality available in FEAP-JS-FW.

Target plattforms

- UNIX platforms:

Cray XT6	(4122 Cores/AMD)	[Production]
Hypercube	(16 Cores/AMD)	[Development/Testing/Processing/Production]
MacBook Pro	(2 Cores/INTEL)	[Development/Testing]

- Compilers:
  - INTEL Compiler Suite
  - GNU Compiler Suite

2.5 Possible Approaches to Parallel Assembly Based on FEAP

In this subsection, we list some possible approaches for the integration of FEAP into an external client code. The possible approaches are then shortly discussed and analyzed with respect to their overall potential. Although only one of the considered options was finally identified as feasible, but most of them were considered as possible starting points at the beginning of the development. This section also gives due credit to previous related work on FEAP interfaces. We start with a list of the considered approaches:

Discussed approaches

1. Extraction of the material library with a new interface.
2. Reimplementation of the material library with a new interface (for example in C++).
3. Customization of an existing FEAP interface.
4. Implementation of a new FEAP interface.

2.5.1 Code Extraction: [rejected]

The idea of this approach is to extract the relevant code sections from FEAP-JS which implement the finite element contribution of the material formulation. At least in principle this code can be encapsulated in a newly designed F77 interface. This new interface should allow to conveniently include the code in a parallel nonlinear solver application.

Code extraction: Advantages vs. Disadvantages

Advantages:

- A flexible software interface which allows for convenient integration of the material models into external codes.
- Light-weight and clean due to elimination of redundant code from FEAP.
- Users have expertise in F77 and already know the code.

Disadvantages:

- The material models use FEAP-internal global data.
- Requires to replace or simulate the basic finite element framework providing the FEAP core infrastructure.
- Testing effort since the extracted components depend on global data.
- Code duplication.

Access to the history variables is one example for a dependency on global data which cannot be avoided for history dependent materials. Extensive use of global data in F77 common blocks and the so-called named fields in FEAP induce a strong coupling between the different parts of FEAP. To give an example, the displacement field U is a global field and many parts of the FEAP code do work on it.

This approach has been investigated by O. Rheinbach and D. Balzani previously (in unpublished work). A specific material model was extracted from FEAP-JS and the extracted code was linked with the FETI-DP-solver. However, this solution had serious drawbacks. The support of new material models necessitated repeated effort for extraction and testing. The models using history

variables for damage were completely unsupported. Thus code extraction was deemed to be not attractive in the long turn and in that sense an unfeasible approach.

2.5.2 Reimplementation: [rejected]

Over the years, J. Schröder himself and multiple doctoral students have contributed algorithms and finite element based material model codes to `FEAP-JS`. The result of these efforts is a quite impressive code base consisting of highly intricate numerical codes. The main idea of the present approach is to reimplement the relevant code sections, preferably in `C++`. This is easiest to interface with the `FETI-DP-solver` code.

Reimplementation: Advantages vs. Disadvantages

Advantages:

- Minimal restrictions regarding the design for the interface. Optimal integration of the material models into external codes.
- Elimination of redundant code leads to a light-weight and clean solution.
- A carefully planned new design might be superior and potent than the current solution (in the long term).

Disadvantages:

- **Incompatible with the current workflow** based on `FEAP` and `F77`.
- Effort to reimplement the relevant material models.
- Effort to test the new implementations.
- Additional design and implementation of a basic finite element framework replacing the heavy dependencies on the `FEAP` core infrastructure.
- A possible performance hit.[a]
- Code duplication.

[a] Implementation of high-performance numerical codes in `C++` is more demanding than in `F77`. It *requires* advanced programming techniques such as expression templates.

finite element assembly of the material library for the planned massively parallel simulations was discarded early on during the planning phase. The usage of automatically generated finite element code in `C++` could however be of future interest Logg, Ølgaard, Rognes, and Wells [92].

2.5.3 Customization of an Available `FEAP`-Interface: [rejected]

A fundamentally different approach is to provide access to an entire instance of the `FEAP` application and to control it externally. Such software exists and is commonly referred to as a `FEAP` interface. All of the solutions mentioned in the following provide interfaces which make the functionality of `FEAP` available to external codes and, ideally, can be used as a basis for a customized interface which meets the identified requirements of our software framework.

Previous FEAP interfaces

Interface	Developer	Institution
MatFEAP	D. Bindel	Cornell University
PyFEAP	C. Y. Chen	Cornell University
pFEAP	M. Adams	Columbia University

The MatFEAP interface Bindel [18] is developed by D. Bindel, currently at Cornell University. It allows to control FEAP from within MATLAB. It is implemented using the MATLAB MEX extension interface. A MATLAB client and a FEAP-server[2] run in different UNIX processes, possibly even on different machines, and use classical UNIX interprocess communication via UNIX sockets or pipes for communication. Thus, by construction MatFEAP did not meet our requirement of a unified shared-memory integrated environment. We acknowledge, however, that we have analyzed MatFEAP during the initial stages of the creation of libfw and that we have drawn some inspiration from its design. For example, the access to variables in FEAP is realized in complete analogy to the way it is done in MatFEAP.

Chen Yi Chen developed a Python frontend called PyFEAP for his masters thesis under the guidance of D. Bindel, see Chen [27]. The frontend client is realized in the programming language python. This is not a unified integration, i.e., the client and the backend do not reside in the same UNIX process.

Mark Adams et al. used FEAP for assembly in the parallel large-scale finite element solver Athena Adams, Bayraktar, Keaveny, and Papadopoulos [1]. In Athena the communication with the sequential instances of FEAP is managed by an interface layer called pFEAP. Presumably pFEAP is quite similar to what has been realized in libfw. It would be quite interesting to compare both approaches. Note that during the development of libfw we were unaware of the existence of pFEAP. Thus, the libfw interface layer was developed completely independently of pFEAP.

We summarize that we have failed to find a previously existing solution which could have served as a basis for our unified parallel solver framework.

2.5.4 Design and Implementation of a New FEAP-Interface: [accepted]

The main idea of this approach is to design and implement a software adaptor, a so-called wrapper, in the form of a software library. This means to turn FEAP into a software library such that the library provides control and data interfaces to an encapsulated FEAP application started in the same UNIX process as the client code.

A first and distinguished advantage of this approach relates to code quality. Only well-tested codes can be assumed to be correct, as is well-known in modern software engineering. The code paths visited by a FEAP wrapper can certainly be expected to be very well-tested, if we assume that the developed interface only uses the official application interfaces. Thus, no bugs are introduced because parts of the FEAP infrastructure have to be reproduced by new code.

This observation also carries over to customized user installations such as FEAP-JS. Note that the testing effort invested by the developers of FEAP-JS into the modeling codes, is applied in a quite specific way. Since the contributions to FEAP-JS are always written as extensions to FEAP, it is only natural that the contributions are only tested *integrated into the FEAP framework* by running the complete application. The downside of this testing strategy is that there is no dedicated unit

[2] MatFEAP provides patches to turn a FEAP installation into a server.

testsuite. A unit testsuite verifies correctness of independent small units of the code, for example, single functions. This, e.g., renders code extraction extremely delicate. However, this is not a disadvantage for the currently discussed approach since it also uses FEAP in its entirety. Moreover, as opposed to the code extraction or reimplementation approaches, in the current scenario, the FEAP-JS *developers can continue to test their own code independently* in the setting they are most familiar with – in standard FEAP. Once their work is tested it can be integrated into FEAP-JS-FW, a particularly extended version of FEAP providing additional wrapper interfaces, with minimal testing effort.

We hope that the reader is convinced at this point, that turning FEAP into a library is at least an interesting option. How can this be realized? On closer inspection, it turns out that the implementation of such a wrapper interface to the entire application is far from obvious. One important aspect is, e.g., the FEAP application state, i.e., the state of the global common data, named fields, open files. All of this should be be properly and consistently managed across multiple calls to the interface. Thus, the entry and exit of the control flow to the FEAP instance must be delicately managed. Let us summarize the advantages and disadvantages of the current approach:

Implementation of a new FEAP interface: Advantages vs. Disadvantages

Advantages:

- Relevant material model implementations are immediately available.
- Interface is consistent with FEAP.
- Most of the FEAP functionality is available including finite element support code.
- Minimal additional testing effort, since the traversed code paths are already tested in FEAP.
- Seamless integration into currently established workflow based on FEAP and F77.
- Fast F77 code.
- No code duplication.

Disadvantages:

- Necessary effort is hard to estimate.
- Possible restrictions due to the integration of a full sequential FEAP instance.
- Need to understand FEAP-internals in detail.
- Testing effort for the wrapper layer.

Conclusion: Our final conclusion on the presented approaches is then as follows: In the given context, the advantages of a reimplementation of the material model codes in C/C++ are clearly outweighed by the disadvantages. The same goes for the approach to extract the relevant parts of the FEAP-JS modelling code. The latter approach was far less realistic than thought of in the beginning, mostly due to the dependencies on the core FEAP infrastructure. To build on a previously available FEAP interface and to tailor it to our needs turned out to be impossible. The options then available to us did not meet our requirements in fundamental ways.The benefits of reusing the available code base of FEAP-JS in a newly developed FEAP interface are striking on first glance – assuming that an interface meeting the specified requirements can actually be

realized. Since the source code of FEAP is distributed to licensed users, a more detailed analysis was to be carried out.

2.6 The Core Component libfw: A New FEAP-Interface

Based on the preceding analysis, the only promising approach turned out to be the implementation of a new FEAP interface. This new interface should ideally turn FEAP into a C-software library complying with the previously formulated requirements for the entire software framemwork. The C-language was chosen, because it is in general convenient to interface with F77 and most other general purpose programming languages. For the realization of the Newton-Krylov-FETI-DP-solver mparfeap, we have implemented such a solution and we shall refer to it as libfw (FEAP-wrapper library).

This section is devoted to the presentation of the public interface of libfw and provides detailed discussions of selected aspects of the actual implementation of the libfw software library. We also try to illustrate software development methodologies which we have found to be valuable during the process of the conversion of the standalone application FEAP into a flexible software library libfw. Although, this is a quite particular task, this might still be inspiring in comparable scenarios.

2.6.1 The Subcomponents: libfw (C) and FEAP-FW (F77)

The software component libfw consists again of two sub-components. The first carries the same name as the entire library and contains the implementation of the public C-interface. This is exposed to the client software. A second component is the internally encapsulated, slightly modified, FEAP code to which we shall refer to as FEAP-FW (F77). These two components are packaged into a single static library called libfw which provides the final adaptor.

The flexible integration of FEAP into possibly quite different parallel simulation frameworks is a nontrivial task. FEAP is a powerful and complex scientific application which was not originally designed to be used as a third-party software library. It was designed to be used as a standalone application. In turn, FEAP does not provide interfaces allowing to modify its control flow or to access internal global data from exterior applications. We shall detail now, our approach to add control flow interfaces to a slightly modified version of FEAP to which we refer as FEAP-FW.

A closer look at the control flow of the unmodified original FEAP application is instructive. Similar to most larger programs, the main routine of FEAP is implemented in the particular form of a so-called event loop. An *event loop* is a simple standard programming construct in computer science which is best illustrated by a short example:

Example 2.6.1. *(A C++ Event Loop) The following code snippet shows an event loop realized in C++. It processes a series of events modelled by an associated Event class. A similar construction is found in the FEAP program control, implemented in F77, where the events are triggered by the FEAP language records parsed from the currently active input stream.*

We note again, that FEAP is set apart from other finite element analysis solutions by its powerful *FEAP command language*. This feature is implemented using an event loop which is located in the program control routine pcontr_.[3] A code which is so flexibly structured that it can be controlled using a scripting language has to be appropriately designed. In other words, this implies certain requirements regarding the code structure. For example, there must be a mapping between command language records to corresponding F77 function calls and parameter sets. Moreover,

[3] The pcontr_ routine is implemented in the file program/pcontr.f.

Example for an event loop in C++

Listing 2.2:

```
int main(int argc, char* argv[])
{
  // Initialization of the program context
  initContext(&argc, &argv);

  // The main event loop
  while(true)
  {
    // Receive next event for processing
    Event event = getNextEvent();

    // Process event and receive event signals
    Signal eventSignal = processEvent(event);

    // Check whether the event signals finalization
    if(eventSignal.isSignaling(EVENT_SIGNAL_FINALIZE))
      break;
  }

  // Finalization of program context
  finalizeContext();
}
```

Figure 2.3: A C++ event loop. Note that the infinite loop can be terminated by any event signaling finalization.

certain contracts regarding the pre- and post-conditions for these calls are to be met, since the ordering of the function calls is determined by the user running his script and is not known in advance.

Each FEAP statement, for example BATCh or TANG,,1, is parsed from the currently active input stream and triggers an associated internal event which is resolved as a function call. Only *the four first letters of a command are relevant* to identify it. The language is case insensitive. E.g., the expressions batc, BATC and Batch are all equivalent and trigger the same event. As depicted in Figure 2.3 *the event loop is entered first after the initialization of the FEAP program context.* The entry point of the FEAP event loop is marked by the F77 jump label 1. This makes it directly accessible from any point in the FEAP program control function pcontr_ using a goto 1 jump statement. This can be used to control the entry of the control flow to the encapsulated FEAP instance such that the FEAP program status can be conserved for subsequent calls to pcontr_. To achieve this, clearly, the initialization code and finalization code sections which clear or at least alter the program state must only be executed when this is necessary. It is only necessary exactly once, during initalization and finalization of the encapsulated FEAP instance, respectively.

2.6.2 Wrapper Interface Extensions for FEAP: FEAP-FW

Turning FEAP into a software library required some changes to the FEAP code. This modified code is referred to as FEAP-FW. We shall now document the most relevant changes made to FEAP.

Controlled Initialization and Finalization

We now turn towards the modifications that allow to control FEAP externally and describe the modified entry behavior, i.e., what happens once the control flow of the client enters FEAP-FW as opposed to what happens in standard FEAP. Note that the input stream is switched to a *simulated input card* in FEAP-FW after initialization. This simulated input card is just a memory buffer which allows to pass FEAP input language records as C-strings to the encapsulated instance. In the current implementation of libfw the program flow and choice of input stream in FEAP-FW are controlled by additional control flags. These flags are declared in FEAP-FW common blocks and can be manipulated from both sides of the C/F77 language barrier and in particular from the C-side of the adaptor. Before FEAP records from the *simulated input card* can be passed to FEAP-FW, it must be initialized. This essentially happens in the same way as in FEAP, by booting the instance from a FEAP input card on the filesystem.[4]

It is important to realize that only the first call to pcontr_ executes the initialization code section in pcontr_ preceding the main event loop in FEAP-FW. On all subsequent calls to pcontr_ the program control flow must skip immediately to the entry point of the FEAP event loop to prevent a reset and reallocation of the FEAP internal data structures. This is achieved by an additional fwjtelflg_ control flag in pcontr_ which is properly set from the libfw C-subcomponent:

FEAP-FW: Controlled entry to the FEAP main event loop

Listing 2.3:

```
c     BEGIN: libfw | Direct jump to 'event loop'.
      if(fwjtelflg) then
c         WRITE(*,*) 'libfw: Jumping to FEAP event loop.'
          goto 1
      endif
c     END: libfw

c     Destroy old output and log files if they exist

      inquire(file=fout,exist=initf)
      if(initf) then
        open (unit=iow,file=fout,status='old')
        close(unit=iow,          status='delete')
      endif

c     Set file for log

      call fileset(fout,flog,'O', 'L')

c     Open files for input, output and log

      open(unit=ior,file=finp,status='old')
      open(unit=iow,file=fout,status='new')
      open(unit=ilg,file=flog,status='new')
      call uscreen(1,iow)
```

Figure 2.4: Code section taken from the start of the pcontr_ function in FEAP-JS-FW. If the fwjtelflg evaluates to .true. the code jumps right to the start of the main event loop (jump label 1) skipping the initialization section of the function. For initialization the fwjtelflg is set to .false. and the initialization code following the if-block is then executed.

[4] This corresponds to the call to initContext() in the example main event loop.

As previously discussed the entry and exit to the FEAP main event loop in the program control must be carefully managed. Let us now dissect the initialization phase of the entire libfw software library. See also the sequence diagram depicted in Figure 2.5.

- Setup of FEAP-FW launch in fwinit_(...):
  - Setup of FEAP-FW control flags for a normal FEAP startup.
  - Setup of standard output streams.
- Start of FEAP-FW session in fw_start_cmdl(...):
  - Export of command line arguments to FEAP-FW.
  - Execution of FEAP-FW application initialization.
  - Execution of FEAP-FW main event loop initialization.
  - Execution of FEAP-FW main event loop.
  - Return from FEAP-FW main event loop (due to a final CNTN in the input card).

To start a FEAP application session, the FEAP executable requires the specification of a valid input card. There are two possibilities to supply FEAP with a filename for an input card:

- Via an interactive startup dialog:
  If started without command line arguments FEAP displays an interactive dialog screen where the name of the input card can be supplied.
- Via a command line switch:
  The command line switch -i allows to pass the name of an input card to FEAP.

For applications on massively parallel remote machines, a non-interactive initialization is certainly the only viable option and we have chosen to use the -i command line switch.

To stay compliant with the major design goal of a flexible highly scalable unified process integration of FEAP with the FETI-DP-solver (or other external client codes) must exchange data via shared memory. To this end both code objects are started in a single UNIX-process. This is possible, but the program launch of FEAP must be slightly altered. Usually, during startup, the command line arguments provided to the executable are passed to FEAP by the operating system when the process is created. This also includes the -i switch which selects the input card to start from in batch mode. A call to the POSIX-commands fork or exec allows to pass command line arguments to a newly created UNIX-process. Bypassing the UNIX-program launcher, the command line arguments can be passed from libfw to FEAP-FW using a shared memory buffer. This is realized through a modified doargs_ function in FEAP-FW that *reads command line arguments from a shared memory buffer*. Realized as a F77 common block, this buffer is initialized by libfw. The startup routine fw_start_cmdl(argc, argv) passes its command line arguments directly to the encapsulated FEAP-FW instance. In case that a -i-switch with a filename for a valid input card is provided, FEAP-FW starts a batch session (non-interactive) from the specified input card.

Let us turn our attention towards the finalization phase. On application finalization libfw essentially executes the same code sequence as FEAP, i.e., the code contained in plstop_. The FEAP-FW finalization procedure fwexit_ is thus derived from plstop_ and is identical with only one notable difference: the control is returned to the original caller and not to the operating system, after finalization. To this end, the final stop statement in plstop_ was simply removed.

To bypass the finalization of the FEAP main event loop, an undocumented CNTN statement is passed to the main event loop for processing. This returns the control flow to the caller of pcontr_ without any alteration of the FEAP program state.

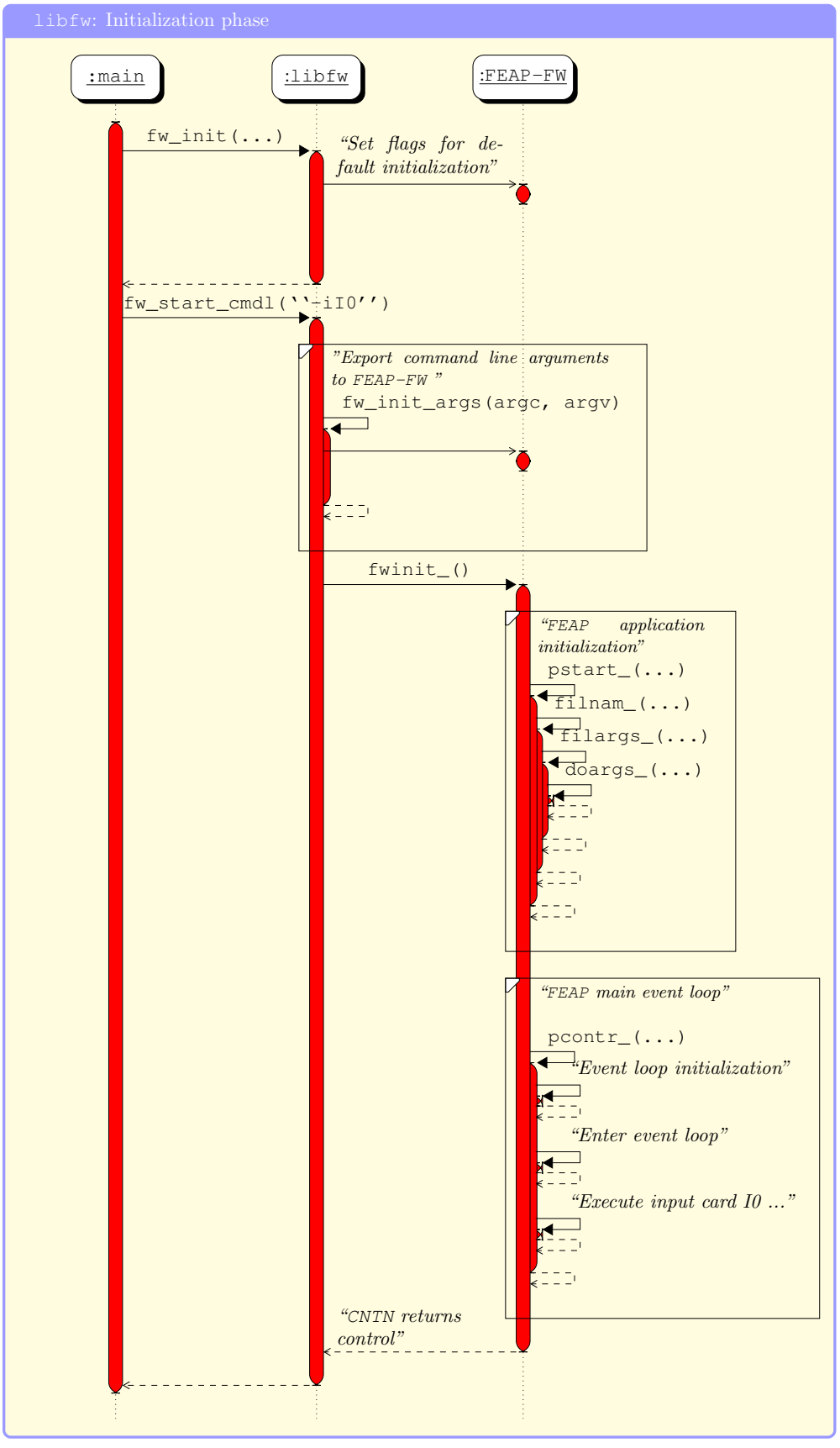


Figure 2.5: Initialization sequence diagram for the FEAP-FW instance encapsulated by libfw.

libfw: Initialization function [fw.c]

Listing 2.4:

```
fw_int fw_init(int* argcPtr, char*** argvPtr)
{

  LOG_INFO("Initializing libfw");
  fw_set_state(FW_STATE_INVALID);
  fw_types_print_info();

  FW_FORTRAN_SYM(fwcomjtel).fwjtelflg = FW_FORTRAN_FALSE;
  LOG_INFO("Initialized shared flag [fwjtelflg = %ld]", \
        (long)FW_FORTRAN_SYM(fwcomjtel).fwjtelflg);

  FW_FORTRAN_SYM(fwcomcio).fwcioflg = FW_FORTRAN_FALSE;
  LOG_INFO("Initialized shared flag [fwcioflg  = %ld]", \
        (long)FW_FORTRAN_SYM(fwcomcio).fwcioflg);

  FW_FORTRAN_SYM(fwcomstate).fwstatesol = 0;
  LOG_INFO("Initialized shared variable [fwstatesol  = %ld]", \
        (long)FW_FORTRAN_SYM(fwcomstate).fwstatesol);

  /* Turn off generation of boundary nodes by default */
  fw_set_flag_pextnd(FW_FORTRAN_FALSE);

  /* Turn off FEAP solver, i.e., activate user solver mode */
  fw_set_flag_solver(FW_FORTRAN_FALSE);

  /* Rewire the output streams for the user */
  fw_io_init();

  LOG_DEBUG("Initialization complete");

  return FW_RC_SUCCESS;
}
```

Figure 2.6: The libfw initialization routine. Printed type information allows to verify libfw and FEAP-FW type compatibility. The setting of fwjtelflg and fwcioflg to .false. effectively initalizes the FEAP-FW instance from an input card file. By default, the calls to pextnd_ are intercepted and user solver mode is deactivated by setting solver = .true..

Remark 2.6.2 (File-based vs. Memory-based Initialization)**.** *In the current implementation of the FEAP-wrapper library libfw, the encapsulated FEAP-FW instance can only by initialized from a FEAP input card file. In the massively parallel setting, this is not ideal. Filesystem usage should always be minimized in an HPC scenario and it would be preferable to initialize the encapsulated FEAP instance from a memory buffer. However, the file-based mechanism is closest to FEAP standard behavior. A full-fledged memory based initialization would also allow to set the FEAP mesh data using shared memory without the current conversion of the mesh data to ASCII-format.*

Modifications for Externalization of FEAP Program Control

We now detail the implementation of the FEAP command interface that allows a libfw client code to execute FEAP command language records. It is part of the libfw public interface. It allows the libfw client code to pass FEAP commands to the wrapped FEAP-FW instance as C-strings. These are then executed by FEAP-FW. For example, a solver might pass the BATCH block shown in Figure 2.9 to FEAP.

libfw: Launch code for encapsulated FEAP-FW instance [fw.c]

Listing 2.5:

```
fw_int fw_start_cmdl(int argc, char* argv[])
{
  LOG_INFO("Launching encapsulated FEAP");

  /* Initializes the command line arguments for feap
   * and passes them via a shared common block (defined
   * in feap/fw/fwcomargs.h).
   */
  fw_init_args(argc, argv);

  /* Launch FEAP */
  fw_io_set_stream_mode(FW_IO_STREAM_MODE_ID_USER);
  FW_FORTRAN_SYM(fwinit)();
  fw_io_set_stream_mode(FW_IO_STREAM_MODE_ID_INITIAL);

  LOG_INFO("Regained control from FEAP event loop");
  fw_set_state(FW_STATE_READY);

  LOG_INFO("Encapsulated FEAP is now ready");

  return FW_RC_SUCCESS;
}
```

Figure 2.7: The `fw_start_cmdl(int argc, char* argv[])` libfw routine launches FEAP -FW with the given command line arguments which are passed via shared memory. An encapsulated FEAP-FW instance is launched by `fwinit_`. **Warning: If the input card selected by the -i-switch does not contain a final `CNTN` statement the call to `fwinit_` does not return.**

Example: A FEAP batch command block

Listing 2.7:

```
BATCh
TANG
FORM
SOLVe
END
```

Figure 2.9: Example for a FEAP batch block. The displayed block consists of five statements. First, the keyword BATCH which opens the batch block puts FEAP into batch mode, where it receives program macros. The first program macro TANG triggers the assembly of the stiffness matrix. The residual is then updated by FORM. Finally the linear solution phase is triggered by SOLV. The block is closed by the final END statement.

This allows us to execute arbitrary blocks of valid FEAP syntax. Let us now dissect the implementation of this mechanism in libfw and FEAP-FW. The necessary control over the FEAP main event loop was already described previously. In FEAP the input is read from one of the standard input streams available in FEAP. In contrast to FEAP, in FEAP-FW this loop is fed with FEAP statements from a simulated memory-resident input card. We have condensed some important

libfw: Finalization procedure [fw.c]

Listing 2.6:

```
fw_int fw_exit(void)
{
   LOG_INFO("Shutting down libfw");

   /* Set internal default assembly callback */
   void* null_ctx_ptr = 0x0;
   fw_set_asm_cb(fw_default_asm_cb);
   fw_set_asm_cb_ctx(null_ctx_ptr);

   /* Set internal default solver callback */
   fw_set_solver_cb(fw_default_solver_cb);
   fw_set_solver_cb_ctx(null_ctx_ptr);

   /* Call the FEAP exit routine to release memory and files */
   LOG_INFO("Passing control to FEAP to let it shut down");
   fw_io_set_stream_mode(FW_IO_STREAM_MODE_ID_USER);
   {
     FW_FORTRAN_SYM(fwexit)();
   }
   fw_io_set_stream_mode(FW_IO_STREAM_MODE_ID_INITIAL);
   LOG_INFO("Reobtained control flow from FEAP");

   /* Flush and reset output streams to the initial settings */
   fw_io_finalize();

   fw_set_state(FW_STATE_INVALID);
   LOG_INFO("Exiting from libfw");

   return FW_RC_SUCCESS;
}
```

Figure 2.8: The libfw finalization code calls the fwexit_() finalization function in FEAP-FW which is derived from plstop_. FEAP-FW then executes the usual FEAP finalization code and flow control returns to the libfw client, not to the operating system via stop_ as in plstop_.

considerations on the externalization of FEAP control flow in Figure 2.10.

Let us revisit the main idea for control flow extraction:

> Turn the FEAP main event loop into a repeatedly callable function and provide the arguments to be executed by the event loop using a simulated FEAP input card.

This function will have side effects since it affects the global state of FEAP. This is even intended, because FEAP state must be preserved across multiple subsequent calls. Via the simulated FEAP input card the user supplies arguments directly to the FEAP event loop which resolves this into internal function calls.

For the realization of the input from shared memory in FEAP-FW, the original FEAP input parsing code in pcontr_ and tinput_() was modified such that it allows to read *input from an internal memory buffer*. This buffer is declared in a F77 common block and can be manipulated from libfw (C) and FEAP-FW (F77).

A verbal description of the sequence of steps executed during execution of a FEAP command in libfw is given in Figure 2.11. It details how FEAP-FW reads input from the shared text buffer

Aspects of the externalization of FEAP flow control

- Controlled initialization of FEAP:
  → Need to pass command line arguments.
  → Control flow must return to caller after initialization.
- Transfer FEAP statements to FEAP for execution:
  → Send statements to FEAP.
  → Receive and execute statements in FEAP main event loop.
- Controlled entry of the FEAP main event loop:
  → Controlled execution of initialization code to sustain FEAP-state across calls.
- Controlled exit of the FEAP event loop:
  → Need to control finalization to sustain FEAP-state across calls.

Figure 2.10: A list of considerations regarding the extraction of the FEAP control flow.

(simulated input card) and executes the contents. A sequence diagram for the same process is displayed in Figure 2.12.

libfw: Verbal description of command execution sequence

1. libfw clears the shared text memory buffer.
2. libfw copies the statements to this buffer.
3. libfw appends a CNTN statement to the buffer:
   → FEAP-FW bypasses finalization and returns control if a CNTN-statement is encountered.
4. libfw sets the shared flag fwjtelflg = .true.:
   → FEAP-FW skips event loop initialization in pcontr_.
5. libfw sets the shared flag fwcioflg = .true.:
   → FEAP-FW reads input from the libfw text buffer using the fwinput_ function.
6. libfw passes control to FEAP-FW with a call to pcontr_.
7. FEAP-FW skips to the start of the FEAP-FW main event loop.
8. FEAP-FW reads the input statements from the shared text memory buffer and executes them.[a]
9. FEAP-FW parses the input from the memory buffer and executes the given input.
10. FEAP-FW executes the appended CNTN command and leaves pcontr returning the flow control to the caller.

[a] Note that FEAP-FW is in control and may call its finalization procedure plstop_ in case of errors, thus terminating the whole application which may be unexpected.

Figure 2.11: Execution steps for the execution of a FEAP command

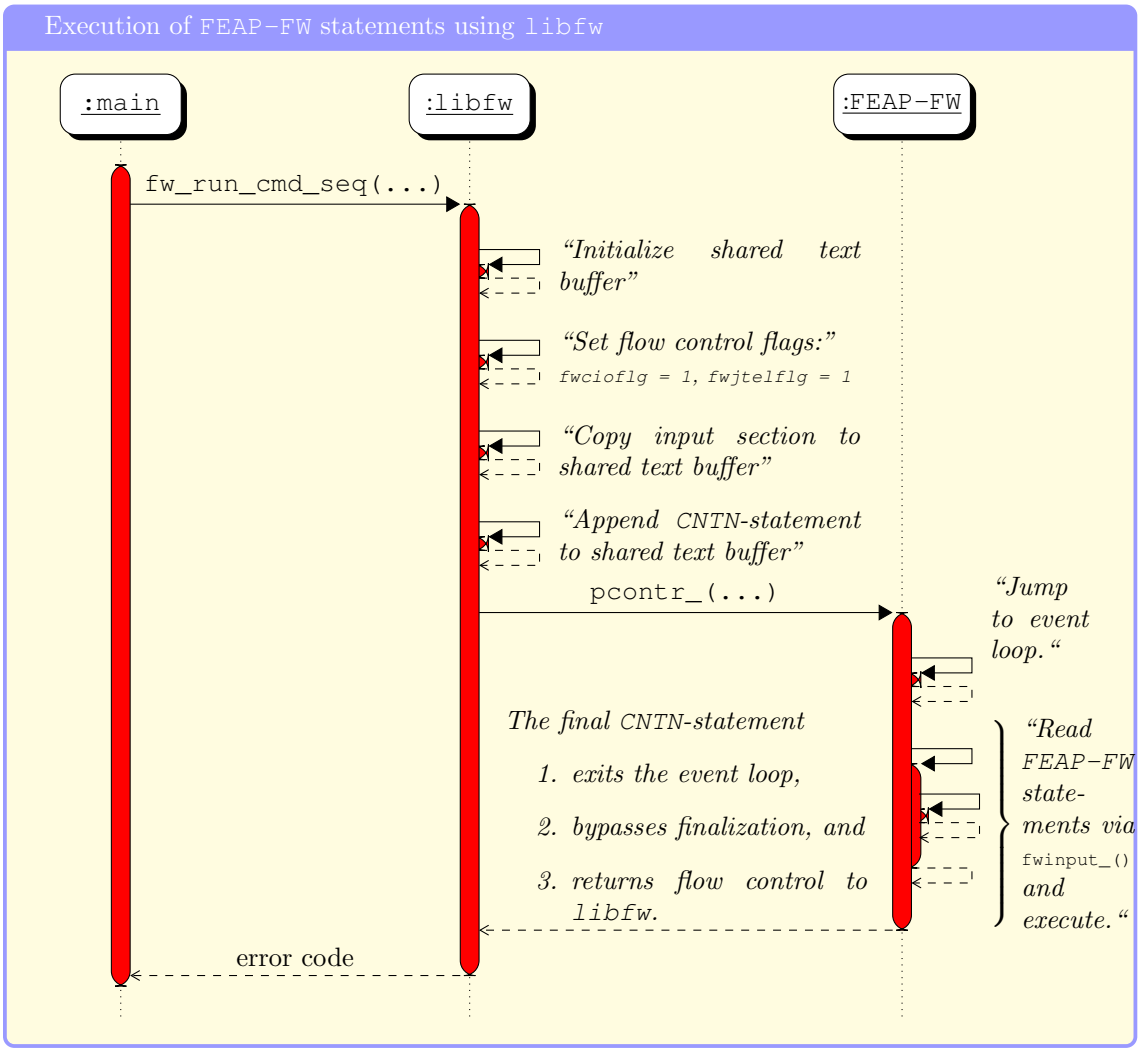


Figure 2.12: Illustration of the event sequence generated by executing a section of FEAP statements using libfw. The FEAP command language statements are read line by line from a shared memory text buffer using a special input function fwinput_. The initialization and finalization code of the FEAP program control function pcontr_ is bypassed to preserve the state of FEAP across consecutive calls to fw_run_cmd_seq.

Modifications to the FEAP Directory Structure

Some files were added to FEAP−FW which are not present in FEAP. The modifications to the directory tree itself are minimal. All files which have been created for FEAP−FW are located in a new subdirectory ./fw. A listing of the additional files together with a short description of their contents is given in Table 2.3.

Modifications to the FEAP Graphics Subsystem

The most important target platform for the intended massively parallel simulations is the Cray XT6 supercomputer at Universität Duisburg-Essen. This and comparable machines are remote accessed e.g., using secure shell access (SSH) and parallel programs are started as batch jobs by a queuing system, e.g., the PBS-system. In this scenario support for X11-graphics is not available. The software library libX11 is not even installed. Consequently, all dependencies on libX11

List of contributed files in the ./fw subdirectory

File Name	Short Description
fw/fwcomm.h	Common blocks for data sharing between FEAP-FW and libfw.
fw/fwexit.f	Exit function for FEAP-FW (derived from plstop).
fw/fwgetpload.f	Function to retreive the proportional loading table.
fw/fwgetvari.f	Function to get (a selection of) variables of type Integer
fw/fwgetvarr.f	Function to get (a selection of) variables of type Real
fw/fwinit.f	FEAP-FW initialization code (derived from feap82.f
fw/fwinput.c	Input function fed from memory buffer (C)
fw/fwsetpload.f	Function to set the proportional loading table.
fw/fwsetvari.f	Function to set (a selection of) variables of type Integer
fw/fwsetvarr.f	Function to set (a selection of) variables of type Real
fw/makefile	Standard directory local makefile from FEAP

Table 2.3: List of the files which have been added in FEAP-FW.

had to be removed and we chose to implement an alternative dummy graphics driver in FEAP-FW. This driver provides the same interface as the FEAP-X11-driver but ignores all received graphics directives.

Using the dummy graphics driver it is possible to use plot commands in libfw also on systems which do not provide X11, without breaking compatibility with input cards that make use of plot commands. This can be helpful since plot commands in FEAP may produce side effects relevant for computations. For testing purposes, we have provided a compile time flag to the libfw build system which allows to select the standard X11 graphics driver. This eases the comparison of results with a FEAP-executable from the stock distribution of FEAP v8.2.

FEAP-FW: Available graphics drivers

- X11 (default)
- Dummy (new)

As previously indicated, the graphics driver can be selected at compile time using a compile time definition in makefile.in, as shown in Figure 2.13.

Modifications to the FEAP Build System

It is instructive to take a closer look at the FEAP build system in order to understand the packaging of the libfw library. The FEAP build system is based on UNIX-makefiles which are configured for the user installation using a makefile.in which is included by all UNIX-makefiles in the FEAP directory tree. When the make tool executes the makefile in the FEAP root directory it executes the following steps:

1. The F77 and C-source files are compiled to object files (with .o-extension). The source subdirectories are visited sequentially.
2. Once generated, the object files for a given subdirectory are automatically *archived* in a static library named feap82.a (by default) using the UNIX archive tool ar.
3. The FEAP main executable program feap82 is compiled and linked with the static library feap82.o.

In the standard distribution of FEAP all external code symbols needed to compile the FEAP main executable feap82 are archived in and provided by the static library archive feap82.a. The build system for FEAP-FW is essentially identical to the FEAP build system. Only the contents of the newly introduced ./fw-subdirectory have to be compiled and archived as well. Similarly, the configuration of the build system has been left as is. I.e., FEAP-FW is configured using a corresponding makefile.in-file in the root directory of the FEAP-FW source tree. There are, however, nonetheless some noteworthy differences between the configuration of FEAP and FEAP-FW. Let us take a more detailed look at the configuration of a FEAP installation. The configuration of a FEAP installation is carried out in the following two files.

1. The build system is configured in makefile.in.
2. The main executable is configured in the program launcher code main/feap82.f.

Let us give a short example for a makefile.in for FEAP-FW and libfw used on MacOSX in Figure 2.13.

FEAP-JS-FW: Configuration in makefile.in (MacOSX/GCC)

Configuration for 64-bit integers. Additional includes for FEAP-JS:

```
24
25 #  FINCLUDE = $(FEAPHOME8_2)/include -I$(FEAPHOME8_2)/include/integer8
```

Selection of the libfw X11 dummy driver:

```
37 FF = mpif90 -fPIC -DLIBFW_USE_X11_DUMMY_DRIVER
38 CC = mpicc -fPIC -DLIBFW_USE_X11_DUMMY_DRIVER
```

Enforce 64-bit width of the INTEGER type (GCC):

```
58 FFOPTFLAG = -ggdb3
59 CCOPTFLAG = -ggdb3
```

Selection of the libfw X11 dummy driver:

```
84   FOPTIONS =
85   COPTIONS = -DLIBFW_USE_X11_DUMMY_DRIVER
```

X11 dependencies have been eliminated and are commented out:

```
91 # LDOPTIONS = -lX11 -lm
92 # LDOPTIONS = -L/usr/X11R6/lib   -lX11 -lm
93 # LDOPTIONS = -L/usr/X11R6/lib   -L/opt/openmpi/lib -lX11 -lm
94 # LDOPTIONS = -L/usr/X11R6/lib64 -lX11 -lm
```

Figure 2.13: A list of relevant configuration settings in makefile.in for FEAP-JS-FW on MacOSX.

Modifications to the FEAP Dynamic Memory Subsystem

Let us now take a closer look at the memory subsystem of FEAP. It is particular for a F77-code, since pure F77 codes can only allocate memory statically. FEAP can allocate memory dynamically through a C memory subsystem contributed by D. Bindel. The so-called *named fields* in FEAP are dynamic arrays with metadata which are managed in a global dictionary. They can be retrieved using the pgetd_ function. Each named field is associated to a dynamically allocated piece of

libfw-js: Example makefile.in (MacOSX/GCC)

```
# Location of FEAP-FW archive
FEAP_FW_ARCHIVE=$(PWD)/../feap-fw-js/libfeapfw-dbg.a
FEAP_FW_ARCHIVE_DIR=$(dir $(FEAP_FW_ARCHIVE))
FEAP_FW_LIBRARY_NAME=feapfw-dbg

#Location of FDDP headers
FDDP_INCLUDE_DIR=$(HOME)/workspace/fddp/src

# Compiler setup
CC  = mpicc
CXX = mpic++

CFLAGS += -DPIC -DUSE_MPI -std=c99 -pedantic -Wall -ggdb3 -fPIC -I$(
    FDDP_INCLUDE_DIR)
#CFLAGS  += -std=c99 -O3 -pedantic -Wall -fPIC -DPIC -DLOG_LEVEL=
    LOG_LEVEL_INFO -DUSE_MPI -I$(FDDP_INCLUDE_DIR)

# Note that FEAP basically uses 2 integer types:
#
# a.) entries of dynamic integer fields (and pointers).
#     (the dynamic integer storage implemented in C.)
#
# b.) standard INTEGER types.
#
# In order to be usable with libfw, FEAP-FW must be
# compiled such that both of these types are of
# the same size, i.e. 4 or 8 bytes.

# 32 bit
#FW_INT_TYPE_WIDTH_IN_BYTES=4

# 64 bit
FW_INT_TYPE_WIDTH_IN_BYTES=8

CFLAGS+=-DFW_INT_TYPE_WIDTH_IN_BYTES=$(FW_INT_TYPE_WIDTH_IN_BYTES)
FFLAGS+=-DFW_INT_TYPE_WIDTH_IN_BYTES=$(FW_INT_TYPE_WIDTH_IN_BYTES)
```

Figure 2.14: Build configuration for `libfw`. Very important is the `FW_INT_TYPE_WIDH_IN_BYTES=8` definition. It indicates consistent use of 64-bit integers in `FEAP-FW`. For this, `FEAP-FW` must be compiled, e.g, with the `-fdefault-integer-8` compiler switch (MacOSX/GCC).

memory which is allocated by a dynamic memory management subsystem (`./unix/cmem.c`) implemented in `C`.

There is a noteworthy technical detail, which any user of `FEAP` or `FEAP-FW` should be aware of when configuring a new installation and we want to mention this at least shortly. For the correct computation of memory offsets in name fields `FEAP` uses a variable called `IPR` (**I**nteger **P**er **R**eal). It is configured in `feap82.f`. A build using 64-bit integer variables requires `IPR = 1` since the `REAL` type is 64-bit wide by default and `IPR = 2` is suitable for 32-bit integers. The build of `libfw` must be configured consistently, see Figure 2.14.

FEAP-FW: Configuration of the IPR-variable

The IPR-variable is configured in the FEAP-FW launch code ./fw/fwinit.f which is executed during the libfw initialization. The FEAP-FW launch code is derived from the original feap executable launcher code in ./program/feap82.f.

We summarize that FEAP uses a dynamic C memory management subsystem. Access to this memory can be realized in FEAP-FW by changing the symbol visibility of the memory pools.

FEAP: Original declaration of dynamic memory pools [mem.c]

Listing 2.8:

```
/* Base address of hr (double) array  */
static byte* mem_system_base_r;
/* Base address of mr (integer) array */
static byte* mem_system_base_i;
```

Figure 2.15: Modificiations to FEAP memory management. The static keyword influences the visibility of the base pointers to the memory pools which are declared here. As originally defined they are only visible from inside the same C translation unit, which is a reasonable access protection. In our context it disallows to reference the memory pool base pointers from other translation units using an extern declaration.

FEAP-FW: Modified declaration of dynamic memory pools [mem.c]

Listing 2.9:

```
/* BEGIN: libfw: | Removed static keyword to render symbols visible. */
/* Base address of hr (double) array */
byte* mem_system_base_r;
/* Base address of mr (integer) array */
byte* mem_system_base_i;
/* END: libfw */
```

Figure 2.16: After removal of the static keyword the memory pools can be accessed from other translation units.

We now turn our focus towards the extension of the user solver interface in FEAP-FW and the corresponding public interfaces in libfw.

2.6.3 Forwarding of User Assembly and User Solver Callback Interfaces

In FEAP and FEAP-FW, the boolean flag solver_ enables or disables FEAP *user solver mode*. By default this flag is set to .false. in the main program routine feap82 at program start. If set to .true., the solution process is delegated by calls to a user solver in multiple phases:

1. The assembly phase: For every element FEAP computes and provides the associated element stiffness matrix and right hand side to the user solver for assembly by calling the user assembly function uasble_ hook.

2. The linear solution phase: FEAP invokes the external user solver at appropriate places in the Newton-scheme via calls to usolve_. Each call also specifies the event type which is encoded by a numeric id.

This is a very flexible mechanism. The implementation of the assembly callback mechanism in libfw is realized based on this as follows: All calls of FEAP-FW to the user solver extension points uasble_ and usolve_ are directly forwarded to the libfw client application together with the arguments provided by FEAP-FW. The libfw client can register callback functions which are invoked whenever FEAP-FW calls uasble_ or usolve_. This is simple, but the additional layer of indirection is sufficient to meet the requirements for a FEAP library.

Thus, in order to realize, e.g., a Newton-solver based on libfw, the client code needs to implements an assembly callback in the C programming language and register this client with libfw. Let us take a look at this mechanism in more detail now.

The libfw Assembly Callback Interface

FEAP assembles elements in an assembly loop. This loop iterates over all elements in the finite element mesh. For each element the element stiffness matrix and right hand side are computed. In user solver mode (solver_ = .false.) FEAP provides the individual element stiffness matrices and right hand sides to the user solver. In particular, the uasble_ function is called for each visited element for the assembly of the linear system which is to be solved by the user solver.

FEAP-FW: Call site of the libfw assembly callback redirector [uasble.f]

Listing 2.10:

```
      call fw_uasble_cb_redirector(s,
     &                             p,
     &                             ld,
     &                             elidx,
     &                             elreg,
     &                             afl,
     &                             bfl,
     &                             ns,
     &                             neq,
     &                             b);
```

Figure 2.17: Call site of the libfw assembly callback redirector in uasble_(). If FEAP is configured for "user solver mode" (usolve_ = 1) the redirection is invoked for each assembled element. The arguments are forwarded to the registered user assembly callback in libfw. In the other mode FEAP-FW uses the FEAP-internal direct solver.

libfw: Assembly call forwarding to libfw [fw_cb.h]

```
void
FW_FORTRAN_SYM(fw_uasble_cb_redirector)(fw_double* K_ptr,
                                        fw_double* f_ptr,
                                        fw_int*    l2g_ptr,
                                        fw_int*    elmt_idx_ptr,
                                        fw_int*    elmt_region_idx_ptr,
                                        fw_bool*   K_asm_flag_ptr,
                                        fw_bool*   f_asm_flag_ptr,
                                        fw_int*    n_elmt_dof_ptr,
                                        fw_int*    n_global_dof_ptr,
                                        fw_double* asm_rhs_ptr)
{ ... }
```

Figure 2.18: The assembly callback redirector in libfw forwards the element assembly events generated by uasble_(...) to the currently registered libfw user assembly callback.

The assembly callback pointer type

Listing 2.11:

```
typedef fw_int (*fw_asm_cb_ptr_t)(fw_double* K_ptr,
                                  fw_double* f_ptr,
                                  fw_int*    l2g_ptr,
                                  fw_int     elmt_idx,
                                  fw_int     elmt_region_idx,
                                  fw_bool    K_asm_flag_ptr,
                                  fw_bool    f_asm_flag_ptr,
                                  fw_int     n_elmt_dof,
                                  fw_int     n_global_dof,
                                  fw_double* cur_rhs_ptr,
                                  void*      user_ctx_ptr);
```

libfw: Assembly callback manager (fw_cb.h)

Listing 2.12:

```
static
fw_asm_cb_mgr_t fw_asm_cb_mgr =
  {
      fw_default_asm_cb, /* register default callback */
      (void*)    0,      /* no user context           */
      (size_t)   0       /* zero call counter         */
  };
```

Figure 2.19: Definition of the static assembly callback manager. By default it is initialized to point to the default user assembly callback fw_default_asm_cb. The default callback is a simple dummy. A call to fw_set_assembly_cb(...) and fw_set_assembly_ctx_ptr(...) stores the current user callback and context pointer here.

libfw: Assembly call forwarding [fw_cb.h]

Listing 2.13:

```
rc = (*fw_asm_cb_mgr.cb_ptr)(K_ptr,
                             f_ptr,
                             l2g_ptr,
                             *elmt_idx_ptr,
                             *elmt_region_idx_ptr,
                             *K_asm_flag_ptr,
                             *f_asm_flag_ptr,
                             *n_elmt_dof_ptr,
                             *n_global_dof_ptr,
                             asm_rhs_ptr,
                             fw_asm_cb_mgr.ctx_ptr);
```

libfw: Public interface for assembly callback registration [fw_cb.h]

Listing 2.14:

```
/** \brief Registers an element assembly callback. */
void fw_set_asm_cb(fw_asm_cb_ptr_t user_cb_ptr);

/** \brief Sets the callback context pointer for the assembly callback. */
void fw_set_asm_cb_ctx(void* user_ctx_ptr);

/** \brief Sets the element assembly callback call counter. */
void fw_set_asm_cb_call_count(size_t count);

/** \brief Returns the currently registered assembly callback. */
fw_asm_cb_ptr_t fw_get_asm_cb(void);

/** \brief Returns the current context pointer for the
     assembly callback.
*/
void* fw_get_asm_cb_ctx(void);

/** \brief Returns the user solver callback call counter. */
size_t fw_get_asm_cb_call_count(void);
```

Figure 2.20: Public interface for assembly callback registration [fw_cb.h].

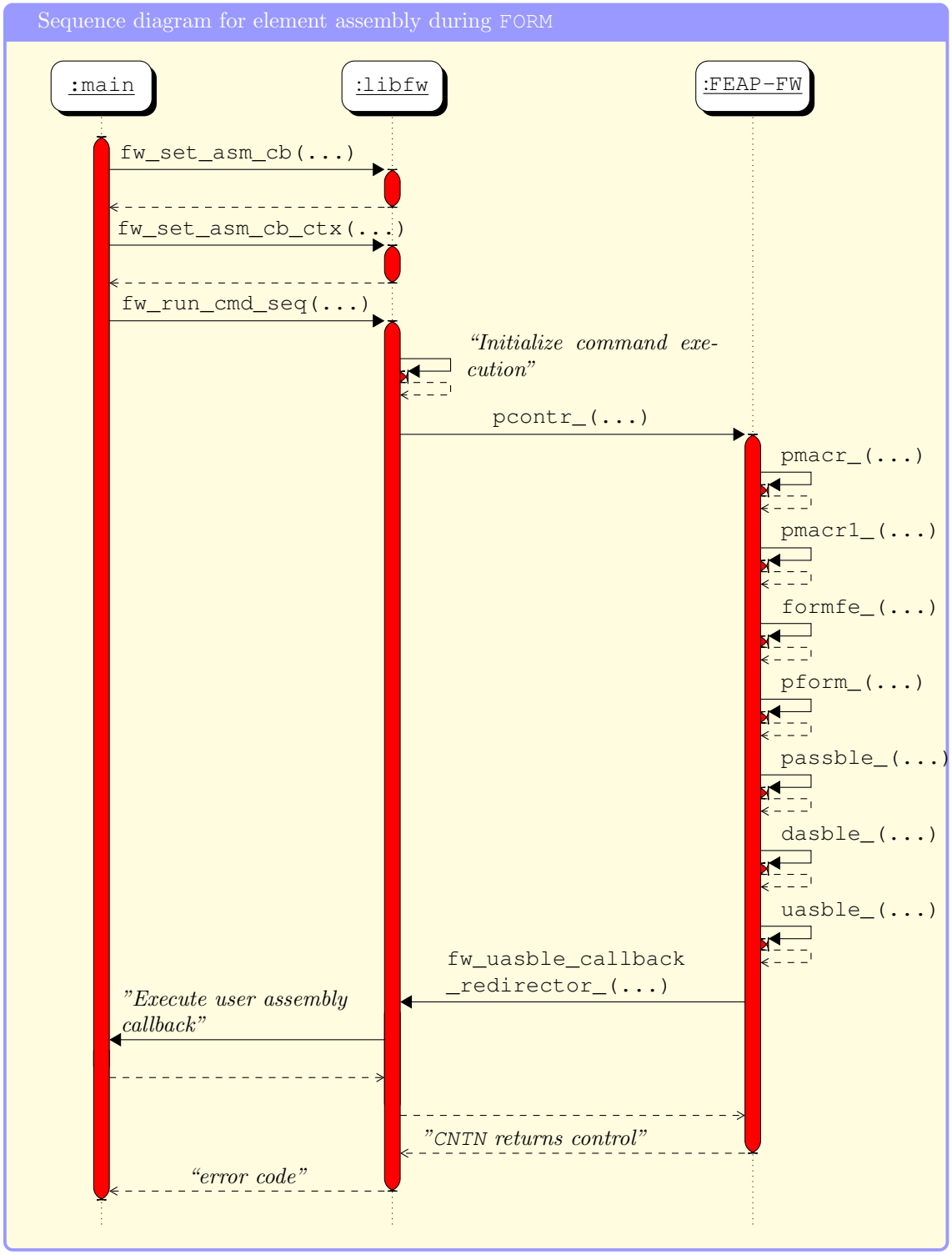


Figure 2.21: Illustration of the event sequence which is triggered by the call `fw_run_cmd_seq` ("BATCH\nFORM\nEND"). The user assembly interface call originitating from `FEAP-FW` is forwarded with one additional level of indirection to the assembly callback currently registered with `libfw`. A client code can register this callback using the `fw_set_asm_cb` function.

	Frame Level	Function Call	Function Definition (File:Line)
mparfeap	#0	Assembly::assembleFEAPElement (this=..., l2g_ptr=..., cur_rhs_ptr=..., user_ctx_ptr)	Assembly.cpp:107
	#1	Assembly::static_assembly_cb_redirector (K_elmt_ptr=..., f_elmt_ptr=..., n_elmt_dof=30, n_global_dof=3018, cur_rhs_ptr=..., user_ctx_ptr)	Assembly.cpp:288
libfw	#2	fw_uasble_cb_redirector_ (K_ptr=..., elmt_idx_ptr=..., n_elmt_dof_ptr=..., asm_rhs_ptr)	fw_cb.c:250
FEAP-JS-FW	#3	uasble_ (s=..., bfl=..., b)	uasble.f:82
	#4	dasble_ (s=..., neqs=..., ad)	dasble.f:131
	#5	passble_ (s=..., al=..., rel=..., jsw)	passble.f:114
	#6	pform_ (ul=..., ie=..., jp=..., nie=..., bfl=..., nn3)	pform.f:386
	#7	formfe_ (pnu=..., bfl=..., nl3)	formfe.f:91
	#8	pmacr1_ (lct=..., _lct=15)	pmacr1.f:786
	#9	pmacr_ (initf)	pmacr.f:503
	#10	pcontr ()	pcontr.f:1523
libfw	#11	fw_run_cmd_seq (cmd_seq=0x1e3bd98 "BATCH \n FORM \n END \n")	fw_cmd.c:158
mparfeap	#12	FEAPControlInterface::run (batchCmd=...)	FEAPControlInterface.cpp:111
	#13	FEAPControlInterface::assembleResidualLocal (assembly=...)	FEAPControlInterface.cpp:207
	#14	FEAPDataInterface::computeResidualLocal (x=..., residualLocal=...)	FEAPDataInterface.cpp:23
	#15	StoredEnergyFEAP::evaluate (this=..., uLocal=...)	StoredEnergy.h:61
	#16	NewtonSolver::solve (this=..., nsci=...)	NewtonSolver.cpp:251
	#17	LoadStepper::computeStep (this=..., nsci=...)	LoadStepper.cpp:90
	#18	LoadStepper::run (this)	LoadStepper.cpp:195
	#19	(argc=24, argv)	Main.cpp:20

Table 2.4: Callstack captured by GDB during element assembly in mparfeap. The FORM command was passed to FEAP in a BATCH block to assemble the nonlinear residual for the next Newton step. Since mparfeap is implemented in C++, the registered callback in libfw is a static function Assembly::static_assembly_cb_redirector redirecting the call to a class-method of the Assembly object. We have stripped the namespace mparfeap here.

The implementation of the forwarding mechanism is realized in a standard way. It follows a simple pattern which allows to set a callback and an additional callback context pointer. This approach is commonly found in C-libraries (for example in OpenGL). It offers support, for use from C++. Note that the context pointer is passed to the callback function on invocation as the last argument.

Remark 2.6.3. *(Usage of the C-callback interface from C++) It is impossible to directly register class methods from C++ as a callback in C. This is a common but small technical obstacle rooting in the fact that* pointers to non-static class methods are not C function pointers. *They can only be resolved relative to the this pointer of an instantiated class instance. Thus, one needs an additional redirection mechanism using an additional static class. The address of a* static class method is a C function pointer. *In this scenario one simply passes the this pointer of the class instance as the callback context pointer fw_set_asm_cb_ctx(this).*

In the following, we give an example how this callback mechanism can be used to register a C++ class method. The user simply supplies the this pointer of the class as the context pointer and implements a suitable static class method which casts the context pointer back to the appropriate class type.

mparfeap: The C++ callback forwarding using a static class method

Listing 2.15:

```
fw_int
Assembly::static_assembly_cb_redirector(fw_double* K_elmt_ptr,
                                        fw_double* f_elmt_ptr,
                                        fw_int*    l2g_ptr,
                                        fw_int     elmt_idx,
                                        fw_int     elmt_region_idx,
                                        fw_bool    K_asm_flag,
                                        fw_bool    f_asm_flag,
                                        fw_int     n_elmt_dof,
                                        fw_int     n_global_dof,
                                        fw_double* cur_rhs_ptr,
                                        void*      user_ctx_ptr)
{
  // Extract this pointer from user_ctx_ptr
  Assembly* assemblyInstancePtr = reinterpret_cast<Assembly*>(user_ctx_ptr);
  ASSERT(assemblyInstancePtr, "Unexpected null pointer!");

  // Redirect the assembly call to the registered instance
  return assemblyInstancePtr->assembleFEAPElement(/* forward arguments */);
}
```

In the context of parallel domain decomposition based linear solvers it is an advantage to have the flexibility to allocate multiple subdomains to a single MPI-instance. It is possible to do this using the previously described implementation. This was also one of the requirements for the parallel framework. To allow for the assembly of multiple subdomain stiffness matrices in a single FEAP instance, we have slightly modified the uasble_ hook function signature. It passes the so-called region index of the currently assembled element as an additional argument to the client code.

Thus, for the realization of the sequential assembly of multiple subdomains per MPI process in mparfeap, every subdomain connectivity is included in a separate FEAP region. This construction allows to relate a given element contribution to a subdomain via the FEAP region index when the uasble_-hook is invoked and forwarded to the registered client assembly function. With

this additional information the element contribution can be assembled into separate subdomain stiffness matrices.

We summarize, that multiple subdomains can be assembled by a single MPI-instance of `libfw` by grouping subdomains in `FEAP` regions.

The `libfw` Solver Callback Interface

The realization of the `libfw` solver callback interface is, for the most part, analogous to the implementation of the assembly callback interface which we have already described in some detail. It suffices here to show the signature of the callback which can be registered in `libfw` using the `fw_set_solver_cb` and `fw_set_solver_cb_ctx` methods. It is important to note that the user solver callback interface provided by `FEAP` is a bi-directional interface, i.e., there is data returned to `FEAP-FW`.

Contract: The Newton correction computed during the linear solution phase must be returned to `FEAP`! Only if this contract is fulfilled, `FEAP` can internally update all fields correctly.

`libfw`: The libfw solver callback pointer type [`fw_cb.h`]

Listing 2.16:

```
typedef fw_int (*fw_solver_cb_ptr_t)(fw_double* rhs_ptr,
                                     fw_bool init_flag_ptr,
                                     fw_bool factor_flag_ptr,
                                     fw_bool solver_flag_ptr,
                                     fw_bool asymmetric_stiffness_flag_ptr,
                                     fw_bool purge_flag_ptr,
                                     fw_int  n_global_dof,
                                     void*   user_ctx_ptr);
```

Figure 2.22: Public type for the `libfw` solver callback which can be registered in `libfw`. Calls to the `uasble_` function are forwarded to the currently registered user solver callback.

libfw: Public interface for solver callback registration [fw_cb.h]

Listing 2.17:

```
/** \brief Sets the current user solver callback */
void fw_set_solver_cb(fw_solver_cb_ptr_t user_cb_ptr);

/** \brief Sets the callback context pointer for the user
    solver callback
*/
void fw_set_solver_cb_ctx(void* user_ctx_ptr);

/** \brief Sets the user solver callback call counter */
void fw_set_solver_cb_call_count(size_t count);

/** \brief Returns the currently registered user solver callback. */
fw_solver_cb_ptr_t fw_get_solver_cb(void);

/** \brief Returns the current context pointer for the
    user solver callback.
*/
void* fw_get_solver_cb_ctx(void);

/** \brief Returns the user solver callback call counter */
size_t fw_get_solver_cb_call_count(void);
```

Figure 2.23: Public interface for solver callback registration in [fw_cb.h].

	Frame Level	Function Call	Function Definition (File:Line)
mparfeap	#0	LinearSolverFETIDP::fwInvocationHookSolve (this=..., user_ctx_ptr)	LinearSolverFETIDP.cpp:74
	#1	LinearSolverFETIDP::fwInvoke (this=..., factor_flag=0, solve_flag=1, asymmetric_flag=1, purge_flag=0, n_dof=3018, user_ctx_ptr)	LinearSolverFETIDP.cpp:43
	#2	LinearSolverFETIDP::static_solver_cb_redirector (rhs_ptr=..., init_flag=0, factor_flag=0, solve_flag=1, asymmetric_flag=1, purge_flag=0, n_dof=3018, user_ctx_ptr)	LinearSolverFETIDP.cpp:120
libfw	#3	fw_usolve_cb_redirector_ (f_ptr=..., factor_flag_ptr=..., n_global_dof_ptr)	fw_cb.c:300
FEAP-JS-FW	#4	usolve_ (flags=..., b)	usolve.f:69
	#5	psolve_ (stype=..., cfr=..., prnt)	psolve.f:204
	#6	pmacr1_ (lct=..., _lct=15)	pmacr1.f:1773
	#7	pmacr_ (initf)	pmacr.f:503
	#8	pcontr_ ()	pcontr.f:1523
libfw	#9	fw_run_cmd_seq (cmd_seq=0x23ce8c8 "BATCH \n SOLVE \n END \n")	fw_cmd.c:158
mparfeap	#10	FEAPControlInterface::run (batchCmd=...)	FEAPControlInterface.cpp:111
	#11	FEAPControlInterface::solveLinearSystem ()	FEAPControlInterface.cpp:223
	#12	LinearSolverFETIDP::solve (this=..., tsci=...)	LinearSolverFETIDP.h:307
	#13	NewtonSolver::solve (this=..., nsci=...)	NewtonSolver.cpp:288
	#14	LoadStepper::computeStep (this=..., nsci=...)	LoadStepper.cpp:90
	#15	LoadStepper::run (this)	LoadStepper.cpp:195
	#16	(argc=24, argv)	Main.cpp:20

Table 2.5: Callstack captured by GDB during the linear solution phase in mparfeap. The SOLVE command is passed to FEAP in a BATCH block. We also see that FEAP passes an event id solve_flag=1 to the currently registered callback function LinearSolverFETIDP::static_solver_cb_redirector. Here, FEAP signals to execute a solution phase.

2.6.4 The libfw Shared Memory Interface for Data Access

Another set of interface methods in libfw is related to read and write access to global data in FEAP. The considered global data in FEAP are of the following two types:

Types of global data in FEAP

- F77 common blocks : statically allocated during program initialization.
- Named fields : dynamically allocated via a C-subsystem.

The previously described minor modification to the FEAP memory management code which changes the visibility of the memory pools is sufficient to implement reliable read and write access to all named fields in FEAP using field information obtained from FEAP by the pdict_ function.

Access to Named Fields

libfw provides interfaces for read and write access to the named fields which are currently managed by the encapsulated FEAP-FW instance. The shared data interface is specified in fw_shared_data.h. All named fields are managed in a global dictionary in FEAP. They can be retrieved using the pgetd_ function which is used by libfw internally. For the sake of completeness, we note that FEAP uses the palloc_ and ualloc_ functions to dynamically allocate memory blocks for named fields internally.

A FEAP named field does not only store data. It also provides the following associated metadata:

FEAP: Metadata for FEAP named fields

Metadata Entry	Type	Short Description
Name	STRING	Dictionary key for the field
ID	INTEGER	Internal integer ID for the field
Precision	INTEGER	Enumeration type for stored data: 0 : INVALID (not initialized) 1 : INTEGER 2 : REAL
Size	INTEGER	Number of stored entries

In what follows, we present the data structures, in particular field handles in Figure 2.25. We have chosen to use handles to the FEAP named fields, in order to implement transparent updates of internal data pointers, for the case that a named field is reallocated in FEAP. This is technically possible and it would lead to stale pointers which might be totally unexpected for the client code.

The access methods for named fields are listed in Figure 2.26.

Access to INTEGER and REAL Variables

Furthermore, libfw provides uniform access to some global variables stored in FEAP common blocks. This functionality is inspired by the solution for variable access in D. Bindel's Matfeap Bindel [18]. A list of supported variables is given in Figure 2.27 and Figure 2.28.

Example: Available named fields with metadata

ID	Name	Length	Type
0	START	0	0
21	JP1	127364	1
25	D	401	2
26	DR	382092	2
27	F	254728	2
28	F0	509456	2
29	FPRO	254728	1
30	FTN	509456	2
31	ID	254728	1
32	IE	13	1
33	IX	1474560	1
34	LD	70	1
35	P	30	2
36	S	200	2
38	T	127364	2
39	TL	10	2
40	U	382092	2
41	UL	140	2
43	X	382092	2
44	XL	30	2
45	ANG	127364	2
46	ANGL	10	2
78	EXTND	127364	1
89	NREN	254728	1
100	RIXT	127364	1
111	TEMP1	127364	1
112	TEMP2	655360	1
151	USER1	11	1
181	RBEN	81920	1
190	NDTYP	127364	1
206	NORMV	382092	2
228	USOL4	127364	2
240	IEDOF	10	1

Figure 2.24: Captured list of *named fields* available during an example run of `libfw`. The shared memory interface of `libfw` provides read and write access to all fields of type `INTEGER` (`type = 1`) or `REAL` (`type = 2`).

Access to the Proportional Loadings

The solution of boundary value problems in nonlinear structural mechanics is usually approached using a load stepping method. `FEAP` offers extensive flexible built-in facilities for load stepping.

In `FEAP`, loading curves can be parametrized using a table of so-called proportional loading factors. This flexible approach allows to model complex loading scenarios. For numerical computations, external access to the `FEAP` proportional loading tables is of interest, since it allows to correctly monitor the loading state of `FEAP`. In a parallel application this eases to infer the correctness. It is quite cumbersome in a massively parallel setting on a remote machine to use the `FEAP` output files for monitoring. Thus `libfw` provides access to the `FEAP` proportional loading tables.

libfw: The field handle data structure

Listing 2.18:

```
/* field handle data structure */
typedef struct
{
  char name[16];
  void* data_ptr;
  size_t length;
  fw_field_type type;
} fw_field_handle;
```

Figure 2.25: The libfw shared memory interface does not expose pointers to the memory location of the data stored in FEAP named fields. Instead libfw uses a field handle data structure. This allows to check for stale data pointers transparently. If needed, the user can easily extract a pointer to the current memory location from the data_ptr member.

libfw: Public interface to named fields [fw_shared_data.h]

Listing 2.19:

```
/** \brief Initializes a field handle to access a named field in FEAP */
fw_int fw_init_field_handle(fw_field_handle* handle_ptr, const char*
    field_name);

/** \brief Internal function to update a field handle if it was relocated
    by FEAP in memory */
fw_uint fw_update_field_handle(fw_field_handle* handle_ptr);

/** \brief Read data from FEAP named integer field */
fw_int fw_get_field_i(fw_field_handle* handle_ptr, fw_int* target_ptr, size_t
    n);

/** \brief Read data from FEAP named double field */
fw_int fw_get_field_r(fw_field_handle* handle_ptr, fw_double* target_ptr,
    size_t n);

/** \brief Write data to FEAP named integer field */
fw_int fw_set_field_i(fw_field_handle* handle_ptr, fw_int* source_ptr, size_t
    n);

/** \brief Write data to FEAP named double field */
fw_int fw_set_field_r(fw_field_handle* handle_ptr, fw_double* source_ptr,
    size_t n);
```

Figure 2.26: The libfw shared memory interface relies on a field handle structure which is transparently updated in case of reallocations of the given named field in FEAP-FW.

Use cases for the access to the loading tables

- External control of the loadings internally applied by FEAP.
- External monitoring of FEAP-internally applied loadings.

The public interface functions in libfw which allow access to the proportional loading tables are

libfw: Available FEAP variables of type INTEGER [fw_shared_data.h]

Listing 2.20:

```
/* accessible FEAP integer variables */
typedef enum
{
  FW_VAR_I_ID_IOR = 0,
  FW_VAR_I_ID_IOW,
  FW_VAR_I_ID_IPR,
  FW_VAR_I_ID_ITECNO,
  FW_VAR_I_ID_MF,
  FW_VAR_I_ID_MQ,
  FW_VAR_I_ID_NADD,
  FW_VAR_I_ID_NDF,
  FW_VAR_I_ID_NDL,
  FW_VAR_I_ID_NDM,
  FW_VAR_I_ID_NEN,
  FW_VAR_I_ID_NEN1,
  FW_VAR_I_ID_NEQ,
  FW_VAR_I_ID_NH1,
  FW_VAR_I_ID_NH2,
  FW_VAR_I_ID_NH3,
  FW_VAR_I_ID_NNEQ,
  FW_VAR_I_ID_NNLM,
  FW_VAR_I_ID_NST,
  FW_VAR_I_ID_NUMEL,
  FW_VAR_I_ID_NUMMAT,
  FW_VAR_I_ID_NUMNP,
  N_FW_VAR_I_IDS
} fw_var_i_id_t;
```

Figure 2.27: A list of integers variables accessible by the shared data interface.

libfw: Available FEAP variables of type REAL [fw_shared_data.h]

Listing 2.21:

```
/* accessible FEAP real variables */
typedef enum
{
  FW_VAR_R_ID_DT = 0,
  FW_VAR_R_ID_TTIM,
  N_FW_VAR_R_IDS
} fw_var_r_id_t;
```

Figure 2.28: A list of real variables accessible by the shared data interface.

listed in Figure 2.30.

2.6.5 Technical Aspects

There are some noteworthy technical considerations and features of `libfw` which we have grouped in this subsection.

libfw: public interface methods for access to variables

Listing 2.22:

```
/** \brief Get value of FEAP integer variable */
fw_int
fw_get_var_i(fw_var_i_id_t var_id, fw_int* value_ptr);

/** \brief Get value of FEAP real variable */
fw_int
fw_get_var_r(fw_var_r_id_t var_id, fw_double* value_ptr);

/** \brief Set value of FEAP integer variable */
fw_int
fw_set_var_i(fw_var_i_id_t var_id, fw_int value);

/** \brief Set value of FEAP real variable */
fw_int
fw_set_var_r(fw_var_r_id_t var_id, fw_double value);
```

Figure 2.29: The libfw shared memory interface relies on a field handle structure which is transparently updated in case of reallocations of the corresponding named field in FEAP-FW.

libfw: Access to proportional loading tables in FEAP [fw_shared_data.h]

Listing 2.23:

```
/** \brief Get current proportional loadings */
void fw_get_pload(fw_double* prop_ptr, fw_int* npld, fw_double* prldv);

/** \brief Set current proportional loadings */
void fw_set_pload(fw_double* prop_ptr, fw_int* npld, fw_double* prldv);
```

Figure 2.30: Public interface for proportional loadings.

I/O Stream Redirection

When used as a library, a user in a massively parallel scenario may want to control the output of the encapsulated FEAP-instance to the STDOUT and STDERR streams. Note that the original FEAP mostly writes output into its logfiles.[5]

To ease debugging and to support custom installations using the standard output streams STDOUT and STDERR libfw provides an interface which allows a user to redirect these standard streams.

A libfw client application may deactivate all output written by FEAP to the standard output streams STDOUT and STDERR by redirecteing them to the /dev/null device during scalability testing. Alternatively, these streams can be routed to dedicated logfiles, for example, indexed by the process number which can be very helpful when tracing down programming errors or unexpected program termination.

[5]This is convenient also for use in massively parallel scenarios. The logfile verbosity can be reduced using the FEAP NOPRint statement.

libfw: Standard output stream setters [fw.h]

Listing 2.24:

```
/** \brief
    Redirect FEAP standard output stream to the given fileno.
*/
fw_int fw_io_set_stream_stdout(int fileno);

/** \brief
    Redirect FEAP standard error output stream to the given fileno.
*/
fw_int fw_io_set_stream_stderr(int fileno);
```

Figure 2.31: Interface to redirect the output streams STDOUT and STDERR to user provided output streams.

The libfw Build Process and Packaging

The build process of the static library archive file libfw.a proceeds in the following three steps. These are executed by the make tool.

1. The FEAP static library feap.a is unarchived using the UNIX ar command line tool.
2. The C source files of libfw are compiled to object files.
3. The original FEAP object files and the libfw object files are archived in a single static library libfw.a.

Thus, we see that the build process of libfw depends on the static FEAP archive, e.g., feap.a. Clearly, the libfw static archive libfw.a can only be built once that archive is available and so build order is important. A complete clean rebuild of libfw.a is accomplished by the following bash-commands:

```
cd $FEAP_FW_HOME          # change to FEAP-FW home directory
make clean && make        # clean tree and rebuild
cd $LIBFW_HOME            # change to libfw home directory
make clean && make        # clean tree and rebuild
```

This packaging strategy has implications for the symbol name space which we summarize in a short remark.

The generated static library libfw.a provides, by construction, the full set of FEAP-FW code symbols. These are visible to any application linked against libfw.

The F77/C Language Border and Type Safety

This section summarizes some of our findings regarding type safety in the given multi-language setting. Interfacing C and F77 is in general not problematic. Some care, however, should be taken when exchanging data between C and F77. Type safety is not guaranteed across C and F77 translation units and the language interface is not entirely specified. There are some technical

subtleties, foremost compiler dependencies, which render it nontrivial to realize a portable multi-language solution.[6]

Let us give an example: the memory layout of members of F77 common blocks, which correspond to C-structs is compiler dependent. More precisely, it may depend on compiler flags, e.g., optimization flags, and pragmas. F77 common blocks can be accessed from a C translation unit. They are usually accessed by declaration of a `extern struct` matching the memory layout. At least in principle, there is the risk of incompatible memory layouts of the same data structure on the C and F77 side due to different compiler flags or even default compiler behavior. Thus, since incompatible packing can – in some cases – lead to erroneous exchange of data between C and F77. To stay on the safe side, in libfw, we have only used common blocks containing a single variable for data sharing. If this is not possible, then data is accessed through getters and setters on the F77 side. This seems to be a robust approach.

Still, type safety stops right at the F77/C-language border. In libfw we have strived for a solution which allows at least the simple reconfiguration of the base types used on the C-side of the interface (libfw) so that they can be matched to the types on the F77 side (FEAP-FW). A corresponding mechanism is realized in `fw_types.h`. The configuration can be influenced at compile time using particular switches defined there.

Error Handling and Signaling

This is one of the aspects that has not been treated in a completely satisfying way. The handling of error conditions is currently only rudimentary. Since FEAP is a standalone application, errors usually terminate the process with an error message. This is a perfectly reasonable behavior for an application, but it is quite inconvenient for a software library. Even more so, for a parallel software library. A software library should pass error states on to its client application which is to decide how to handle a given error state.

Let us give an example for the currently implemented error handling. The nonlinear solver mparfeap implements a simple adaptive load-stepping strategy using libfw. It is necessary to detect invalid deformed configurations. Here, by invalid we mean that the material code fails to correctly compute the contribution for at least one finite element. Previously, in FEAP-JS, the code just called `plstop_()` and terminated the program with an error message, if an error was detected during element assembly. This is not fault tolerant and did not allow for the implementation of load-stepping. To overcome this, a simple signal mechanism was implemented to detect, e.g, invalid tangents. Currently this mechanism is only implemented in a rudimentary way and allows to pass error codes from FEAP-FW to the client indicating a current status integer-id.

libfw: Signaling mechanism for invalid solution states [fw.h]

Listing 2.25:

```
/** \brief Get the status indicator for the last solution. */
fw_int fw_get_state_solution();

/** \brief Get the status indicator for the current solution. */
fw_int fw_set_state_solution(fw_int state);
```

[6] Portability (among UNIX platforms) is clearly an important requirement here, since code based on libfw may be run on different supercomputers, for example, for scalability testing of linear and nonlinear solvers based upon the library.

This is specifically tailored to the adaptive load-stepping implemented in `mparfeap` and for `libfw-js`.

The addition of a convenient interface allowing to signal errors in `FEAP-FW` to a `libfw` client code is of interest, but seems not trivial to realize. This might be future work.

Reliable Tracking and Runtime Analysis of Modifications

It was an explicit request of the `FEAP-JS` developers, and also fixed as part of the requirements definition, that all modifications done to `FEAP` (or `FEAP-JS`) are documented in the source code of `FEAP-JS-FW` so that modified code sections can be identified on first glance. This is an error prone task. To assure this, a **S**ource **C**ontrol **M**anagement (SCM) software was used to track the changes done to `FEAP` v8.2.[7] The typical advantages of using an SCM tool for software development apply. Let us single out two particular advantages which are of immediate importance in the current scenario:

1. *All* modifications done to the stock distribution of `FEAP` v8.2 can be retraced (with metainfo on changesets).
2. A UNIX software patch in `diff`-format containing all differences between `FEAP-FW` and the repository base revision `FEAP` v8.2 can easily be generated for distribution.

A UNIX patch allows to the migration to newer versions or other custom user installations of `FEAP` based on `FEAP` v8.2. Such management of the code modifications might well prove to be crucial for the future maintenance of the `FEAP` wrapper code basis. This is just one of multiple means to cope with the given scenario trying to implement good software practices, for example, most importantly to ease maintainability.

Modification tracking using an RCS (**R**evision **C**ontrol **S**ystem): To ease the process of locating code sections in `FEAP-FW` (and `FEAP-JS-FW`) which introduce modifications to the base revision, the modified code sections were marked. Special code markers bracket these changes using particular `F77` comments. This is very helpful in practice, as it is easy to locate modified code sections using, for example, `grep 'libfw'`. A short example for such a modification in `FEAP-JS` is depicted in Figure 2.32, where a new header file contributed to `FEAP-FW` is included and marked as a change.

`FEAP-JS-FW`: Modification markers [`js_elements/elmt_dobr15.f`]

Listing 2.26:

```
c.....BEGIN: libfw
      include '../fw/fwcomm.h'
c.....END: libfw
```

Figure 2.32: Modified code section enclosed by the code markers `BEGIN: libfw` and `END: libfw`. A developer workin on `FEAP-JS-FW` can easily identify the modified code section and infer that the include directive was modified or newly introduced in `FEAP-JS-FW`.

This approach gives the developer the information that the code section framed by modification markers was modified. However, only the current status of the code modification, i.e., the resulting

[7] In the beginning the Mercurial SCM tool (`hg`) was used to keep track of modifications, then we migrated to `git`.

code, is visible to the developer. Additional information is needed to deduce exact source code modifications. This information can be retrieved from the git-repository in the form of code differences in the standard diff format. The base revision of the FEAP-FW git-repository FEAP -FW is the standard distribution of FEAP v8.2. Using git (or mercurial) helps to ascertain that *all modifications done to FEAP v8.2 can be retraced* and documented. Clearly, the complete listing of code differences between FEAP-FW and the standard distribution of FEAP v8.2 renders it possible to identify modified sections where code markers are missing.

Besides intensive support on FEAP details from the FEAP-JS developers, in particular from D. Brands, for the analysis of the impact of access to FEAP through the libfw package, we have applied some runtime code analysis. As FEAP is not a small software package this proved indeed helpful. It is nontrivial to analyze efficiently how a given part of the FEAP code operates internally at runtime. Usage of a debugger such as gdb can be inconvenient to understand the generated application callgraphs, since it only produces a static view. To quickly understand callgraphs of FEAP triggered by libfw events, the etrace-package was used, see Vanegue, Garnier, Auto, Roy, and Lesniak [136]. During the initial phase of the development of FEAP-FW this analysis was very helpful to gain an understanding of the internals of FEAP. etrace uses compile time instrumentation of the function entry and exit points to record the call stacks produced during application lifetime. This was also helpful in order to verify that the modifications done to FEAP in FEAP-FW do produce the expected control flow.

The libfw wrapper library is a new type of FEAP-interface. Its use is not limited to the currently presented context, i.e., to the assembly of models available in FEAP-JS. It is also not limited to mparfeap. It has also been independently used for the assembly in fluid-structure simulations in LifeV.

2.7 The FEAP Domain Decomposition Processor: fddp and libfddp

Any domain decomposition-based approach and in particular FETI-DP requires that the global computational domain be decomposed into subdomains. In the present framework, the geometry and the boundary data are decomposed in a sequential preprocessing step. The input data for this step are FEAP input cards including geometry data by a custom command line application called fddp (**F**EAP **D**omain **D**ecomposition **P**rocessor). This application is implemented on top of an associated utility library called libfddp. A basic, extensible object oriented parser for the FEAP input card format written in C++ allows to read FEAP problem descriptions. Parts of the code in libfddp are also directly reused by the parallel solver mparfeap. This eases the implementation of consistent data handling between the preprocessor fddp and the parallel solver backend mparfeap.

The fddp tool is capable of computing primal and dual subspaces for FETI-DP for unstructured as well as for structured domain decompositions after the mesh partioning process.

As part of the input data sets for mparfeap, the fddp command line tool generates ASCII-files describing the generated FETI-DP primal and dual subspaces. These spaces are characterized by a list of generators, i.e., a spanning set of the associated subspace. The local primal and dual spaces are stored in one .lps-file (primal space) and one .lds-file (dual space) per FETI-DP subdomain. The format is the same for both files. Every line describes a single generator by: a type id, a local entity index, a component index.

The currently supported entity types are: nodal values, edge averages and face averages. Depending on the generator type, the associated entity index refers to a node, an edge or a face. Edge and faces are internally identified with the first node contained in the set. The implementation of

edge and face averages is based on the transformation of basis approach, see e.g., Rheinbach [110]. Note that the degrees of freedom associated to the components of the displacement vector field $u : \Omega \to \mathbb{R}^3$ are handled separately. This allows, e.g., to impose the sliding boundary conditions used by Brands, Klawonn, Rheinbach, and Schröder [23], by Brinkhues, Klawonn, Rheinbach, and Schröder [25] and also in the high resolution calculations of arterial wall stresses presented in Chapter 3.

We summarize that the input of the mparfeap solver to be presented in the following section is generated by the sequential fddp preprocessor.

2.8 The Newton-Krylov-FETI-DP Solver: mparfeap

In this section, we present a Newton-Krylov-FETI-DP solver called mparfeap (**M**assively **P**arallel **F**inite **E**lement **A**nalysis **P**rogram) allowing to run massively parallel simulations based on FEAP -JS and the FETI-DP-solver due to Oliver Rheinbach. So far, we have only discussed some underlying software components for this integration.

This is a unified shared-memory integration of FEAP-JS with the FETI-DP-solver (corresponding to integration stage IV) allowing for the simulation of arterial wall structures, which was the original motivation. The mparfeap solver is implemented in C++ and uses libfw-js (i.e., it wraps FEAP-JS) for the assembly of the linear systems arising in the Newton algorithm. Once assembled, these linear systems are transformed into a FETI-DP master system and solved using the scalable FETI-DP-solver. The Newton correction in each Newton step is thus computed using FETI-DP. Since a direct application of Newton's algorithm is in general impossible for hard mechanical problems as those presented here, mparfeap provides load stepping facilities. Classical load stepping is a homotopy method for the loadings and is also a globalization strategy for the Newton's algorithm.

The input for mparfeap is computed by the previously introduced fddp application. This preprocessor generates the FEAP input cards from which the libfw-js MPI-instances are initialized.

2.8.1 Input Data

Let us give an example for an input data set for mparfeap. It was generated by the fddp preprocessor and consists of four worksets allocated to two MPI-processes.

```
I0
I0_0v
I0_1v
I0c
I1
I1_0v
I1_1v
I1c
```

Figure 2.33: FEAP input cards generated for a parallel run by the fddp; 2 MPI processes (leading index); 2 local meshes (second index) per MPI-process; local node coordinates are stored in a I*c-file; local volume element lists are stored in a I*_*v-file

```
0.wbi
0_0.dbc
0_0.lds
0_0.lps
```

Figure 2.34: The `0.wbi` is a workset bundle info. It contains metadata on the global and local problem size. Dirichlet boundary conditions are stored in the `0_0.dbc` file. The local degrees of freedom in the FETI-DP primal space are encoded in a `.lps`-file and the local dual space degrees of freedom in a `.lds`-file.

```
local_corner_2_local_nodes_0_0
local_corner_2_local_nodes_0_1
local_edge_2_local_nodes_0_0
local_edge_2_local_nodes_0_1
local_face_2_local_nodes_0_0
local_face_2_local_nodes_0_1
local_node_2_global_domains_0_0
local_node_2_global_domains_0_1
local_node_2_global_node_0_0
local_node_2_global_node_0_1
```

Figure 2.35: Geometry information used to build the linear systems for the `FETI-DP-solver` integrated into `mparfeap`. The primal space (coarse component) for FETI-DP is read from local files with a `.lps` extension. These input files also provide information to compute edge and face averages in the transformation of basis approach.

```
FetiDP_Bbr_0_0.mat
FetiDP_Bbr_0_1.mat
FetiDP_Bc_0_0.mat
FetiDP_Bc_0_1.mat
FetiDP_BrT_0_0.mat
FetiDP_BrT_0_1.mat
FetiDP_rho_local_0_0.vec
FetiDP_rho_local_0_1.vec
```

Figure 2.36: Sparse `PETSc`-matrices and The FETI-DP jump operators depend only on the connectivity of the subdomain interface. They are precomputed by the `fddp` in a sequential step.

2.8.2 Parallel Assembly

One of the main requirements and features of the presented developed framework is MPI parallel assembly of the subdomain stiffness matrices for the FETI-DP solver.

Indexing of Degrees of Freedom in `FEAP`

Multiple element connectivity lists can be included by a single `FEAP` input card. The element connectivity lists correspond to volume or surface meshes. Each mesh can be imported into a different `FEAP` region. This is accomplished using the `REGION` command. Internally, the `REGIon`

command simply assigns an integer region number to all elements in a finite element volume or surface mesh which are contained in the given region. Note that FEAP is restricted to one coordinate list per input card. This list is shared by all finite element nodes. Let us describe the degree of freedom mapping used by FEAP in uasble.f as it is stored in memory (and seen from C). It is node-based, e.g., in three dimensions the 0-based mapping for the degrees of freedom is given by

$$\mathrm{dof}(i,j) \quad = \quad 3\,i + j\;.$$

Here, i denotes a 0-based node index and j a 0-based component index. Since FEAP is a F77 code, the indexing in the FEAP program code is 1-based, but we access the raw data in shared memory using 0-based C-indexing.

Sequential Assembly of Multiple Subdomains per MPI Process

Parallel scalability on large machines requires that the domain partitioning is flexible, to some degree. This allows to optimize the memory profile of the solver application which is essential in practice. In mparfeap multiple subdomains can be managed in a single MPI-instance. This is realized using the FEAP region command REGIon in the input data sets to group multiple subdomains in one input card per rank. All subdomains which are managed by a single MPI-instance are assembled sequentially.

Finite element meshes can be grouped into FEAP regions. This allows to assemble multiple subdomains in a single FEAP input card. These are assembled sequentially into one subdomain stiffness matrix per FEAP region.

Let $0 \leq r \leq r_{\max}$ denote the rank of the current MPI process and suppose $N_r \in \mathbb{N}$ subdomains are associated to the r-th FEAP-instance via the REGIon command. The FEAP node coordinate list generated by the fddp is obtained by simply concatenating the subdomain node lists while enforcing a consecutive node numbering. The element lists are split into one element list per FEAP region referencing the nodes local to the subdomain.

With these input data FEAP sets up the local-to-global mappings to assemble the subdomain stiffnesses into a direct sum of the stiffness operators which are assembled by the process, i.e.,

$$K^{(r)} = \bigoplus_{j=1}^{N_r} K_j^{(r)}, \quad \forall r = 1, \ldots, r_{\max}$$

by default. Here, we have denoted the stiffness matrix assembled for the j-th FEAP region in the r-th MPI process by $K_j^{(r)}$. The handling of the right hand sides is completely analogous. Note that internally FEAP uses a 1-based interleaved mapping for the degrees of freedom. Since FEAP passes the element stiffnesses and right hand sides to the user via the user assembly interface (uasble.f) the user has full control over the final assembly layout. For example, in mparfeap, we assemble into separate matrices, i.e., into a cartesian product

$$\left(K_1^{(r)}, \ldots, K_{N_r}^{(r)}\right)\;, \quad \forall r = 1, \ldots, r_{\mathrm{rmax}}$$

and use 0-based indexing for the degrees of freedom.

We conclude with a short example input card for the case of four subdomains generated by the fddp application, see Figure 2.38. These four regions are assembled using a single sequential MPI-instance of FEAP in mparfeap.

mparfeap: Assembly routine for FEAP element data

Listing 2.27:

```
// Assemble FEAP element contribution
fw_int
Assembly::assembleFEAPElement(fw_double* K_elmt_ptr,
                              fw_double* f_elmt_ptr,
                              fw_int*    l2g_ptr,
                              fw_int     elmt_idx,
                              fw_int     elmt_region_idx,
                              fw_bool    K_asm_flag,
                              fw_bool    f_asm_flag,
                              fw_int     n_elmt_dof,
                              fw_int     n_global_dof,
                              fw_double* cur_rhs_ptr,
                              void*      user_ctx_ptr)
{
  // SNIP: Check for valid arguments
  LOG_DEBUG(std::string("Assembling FEAP element ")
            + "[region = "  + to_string(elmt_region_idx) + "]"
            + "[element = " + to_string(elmt_idx) + "]");

  int dofOffset = dofOffsets_.at(elmt_region_idx);
  if(K_asm_flag)
    {
      LOG_DEBUG(std::string("Assembling element stiffness matrix ")
                + "[region = " + to_string(elmt_region_idx) + "]"
                + "[element = " + to_string(elmt_idx) + "]");

      MapMat<double>& K = stiffnessMatrices_.at(elmt_region_idx);
      for(int i = 0; i < n_elmt_dof; ++i)
        for(int j = 0; j < n_elmt_dof; ++j)
          {
            // Subtract region dof offset and convert fortran 1-based index to
                 0-based
            int iA = (l2g_ptr[i] - 1) - dofOffset;
            // Range check: If the entry cannot be assembled just continue
            if(iA < 0) continue;

            // Subtract region dof offset and convert fortran 1-based index to
                 0-based
            int jA = (l2g_ptr[j] - 1) - dofOffset;
            // If the entry cannot be assembled just continue
            if(jA < 0) continue;

            // All idxs in range -> Element can be assembled.
            LOG_DEBUG("Assembling K: [" + to_string(i) + "," + to_string(j)
                      + "] -> [" + to_string(iA) + "," + to_string(jA) + "]");

            // Note: i and j must be interchanged on the RHS due to FORTRAN
            //       column major address calculation
            const double& v = K_elmt_ptr[j * n_elmt_dof + i];
            K(iA, jA) += v;
          }
    }

  // SNIP: Assembly of right hand side
  return 0;
}
```

Figure 2.37: The element region index is used to assemble the element contribution into the corresponding subdomain stiffness matrix. Note also the F77 indexing (1-based) and memory layout (column-major) conventions.

fddp: Generated FEAP input card with 4 subdomains

Listing 2.28:

```
FEAP    ! Automatically generated by FDDP
0,12,1,3,3,10

REGI,0
ELEM
INCL,I0_0v

REGI,1
ELEM
INCL,I0_1v

REGI,2
ELEM
INCL,I0_2v

REGI,3
ELEM
INCL,I0_3v

COOR
INCL,I0c

MATErial 1
SOLId
ELAStic STVK 210E9 0.324086

END

CNTN
```

Figure 2.38: A FEAP input card generated for mparfeap by the fddp preprocessor. 4 subdomains are allocated to a single MPI-instance. The subdomain meshes are imported into separate FEAP -regions. Dirichlet boundary conditions are filtered out by the fddp. The final CNTN-statement returns control from FEAP-FW to libfw.

2.8.3 The Solver Class Hierarchy

In the mparfeap solver, the outermost loop which controls the loadings, i.e., the Dirichlet and Neumann boundary conditions are implemented in the mparfeap::LoadStepper class. This class currently supports two different loading strategies: linear and adaptive. A standard Newton algorithm is implemented in the mparfeap::NewtonSolver class. Different convergence criteria can be selected. For the computation of the Newton-corrections in the Newton algorithm the highly scalable FETI-DP-solver by O. Rheinbach is used and integrated using a mparfeap::LinearSolverFETIDP class. This class extends the abstract base class mparfeap ::LinearSolverAbstractBase which is in the mparfeap framework.

The next chapter presents massively parallel scalable simulations for applications in the biomechanics of biological soft tissue models computed with the massively MPI-parallel Newton-Krylov-FETI-DP solver mparfeap.

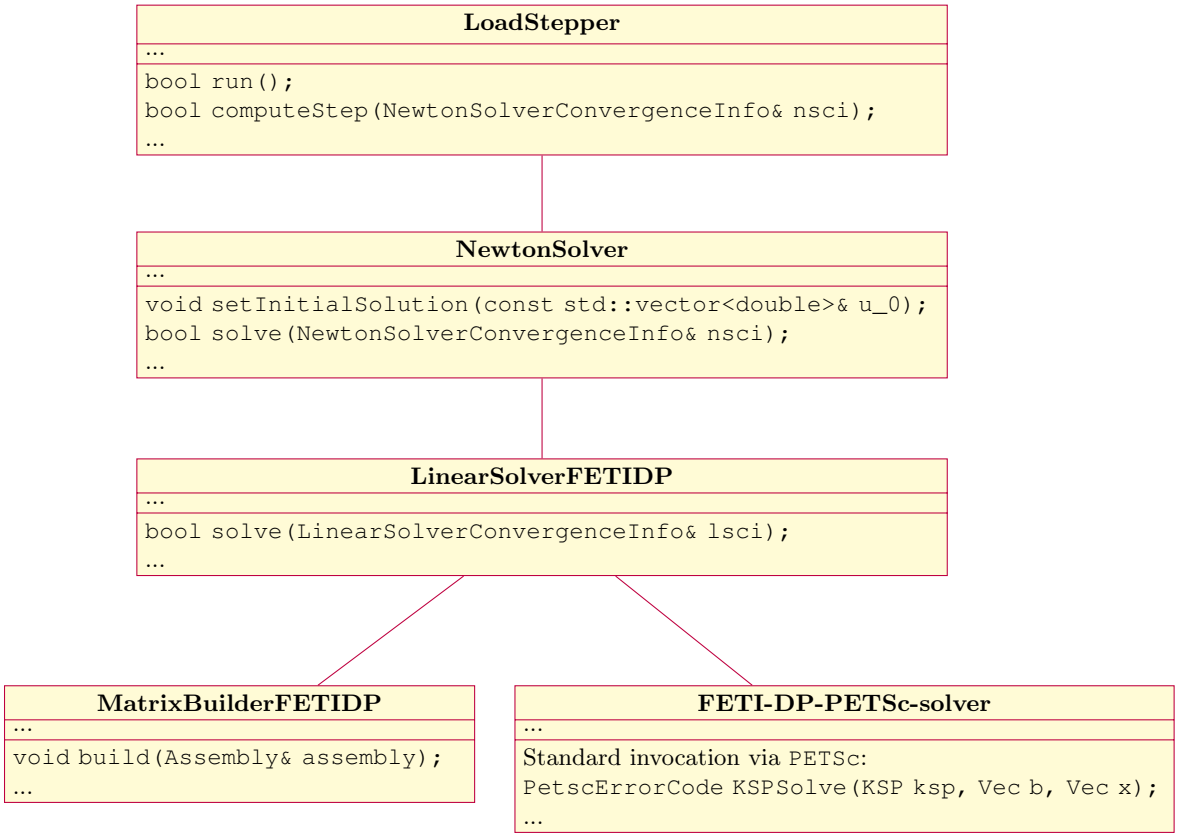


Figure 2.39: Nesting of solver classes in `mparfeap`. The load stepping homotopy method is implemented in the `LoadStepper` class. After every load increment a nonlinear problem is solved by the `NewtonSolver` class. Newton corrections are computed using FETI-DP by a linear solver class `LinearSolverFETIDP`. This class initializes and executes the `PETSc`-based `FETI-DP-solver` developed by O. Rheinbach. The system and preconditioner matrices for the `FETI-DP-solver` are set up by the `MatrixBuilderFETIDP` class.

mparfeap: Hook for the linear solve action from FEAP-JS-FW

Listing 2.29:

```
/* Solve the linear system */
void LinearSolverFETIDP::fwInvocationHookSolve(fw_double* rhs_ptr,
                                               fw_int n_dof,
                                               void* user_ctx_ptr)
{
  LOG_DEBUG("Invoking tangent solver now [n_dof=" + to_string(n_dof) + "]");

  solveCallBack();

  if(primalSolutionLocalJoined_.empty())
    {
      LOG_DEBUG("Primal solution is not available. Passing du = 0 to FEAP");
      primalSolutionLocalJoined_.resize(n_dof);
    };

  size_t sz = primalSolutionLocalJoined_.size();
  ASSERT(sz == static_cast<size_t>(n_dof),
         "Newton update du has wrong dimension [" + to_string(sz) + "]");

  // CONTRACT with FEAP (usolve.f):
  // The solution must be copied back to FEAP for the update of the
  // solution in FEAP.
  LOG_DEBUG("Copying Newton update back to encapsulated FEAP instance");

  ASSERT(math::checkArray(primalSolutionLocalJoined_), "Numerical garbage");
  memcpy(rhs_ptr, &primalSolutionLocalJoined_[0], n_dof * sizeof(fw_double));
}
```

Figure 2.40: Implementation of the hook which is invoked when FEAP-JS-FW calls usolve_ for the solution of the linearized system. The FETI-DP system is built and solved in solveCallBack and the primal solution is passed back to FEAP-JS-FW in the right hand side using memcpy. This contract is part of the usolve-interface definition. It is used to update the FEAP-internal copy of the solution vector and other internal data structures such as the material history after every Newton-step.

CHAPTER 3

Applications

This final chapter presents a selection of large-scale numerical computations for soft biological tissues in biomechanics obtained using the Newton-Krylov-FETI-DP solver `mparfeap` on a Cray XT6m supercomputer provided by the **C**enter for **C**omputational **S**ciences and **S**imulation (CCSS) at Universität Duisburg-Essen.

3.1 Overview

We shall try here to condense the essential aspects of the present work into a good starting position for the interpretation of the parallel simulation results presented in the following sections. A more extensive general introduction can be found in Chapter 1, a description of the parallel software framework is the theme of Chapter 2. There also the terminology for the software components used here is defined. Note also the supplementary Appendix A which provides a non-technical introduction to the framework of anisotropic hyperelasticity.

The text at hand has its origin in the interdisciplinary research project

Massiv parallele Simulationen von Arterienwänden: Kontinuumsmechanische Modellbildung und numerische Lösung mittels FETI-Gebietszerlegungsverfahren[a]

(KL 2094/1-{1,2}, SCHR 570/7-{1,2})[b]

[a] Massively parallel simulations of arterial walls: Continuum mechanical modeling and numerical solution via FETI domain decomposition methods (title translated by the author).

[b] The project was supported by the Deutsche Forschungsgemeinschaft (DFG) in two funding periods; common grant proposal by PIs Axel Klawonn and Jörg Schröder.

A highly fruitful collaboration that spawned an entire series of publications [12, 13, 21–23, 25, 52, 73] and previous dissertations [20, 24, 55] which are foundational to the simulation results here obtained. We show results for:

§3.2 High-resolution computations of transmural von Mises-stress distributions using patient-specific atherosclerotic arterial wall models subject to a maximal interior normal pressure of 500 [mmHg].

§3.3 Strong scalability results for these aforementioned high-resolution computations of transmural wall stresses.

§3.4 Weak scalability results for the structured decompositions of model problems on cuboidal domains using up to 4096 CPU cores.

§3.5 Two simple tuning strategies for a Newton-Krylov-FETI-DP solver nested in a load stepping scheme:

- Initialization of the inner linear FETI-DP solver with the last successful dual linear solution $\lambda^{(n-1)}$.
- Linear extrapolation of the displacement u to compute the initial value for the Newton iteration in the next loading attempt.

We recall from §1.3 that FETI-DP is a non-overlapping domain decomposition method which is characterized by a choice of a primal coarse space. This coarse space is induced by a selection of certain continuity constraints. The nested FETI-DP solver in `mparfeap` is denoted here as the `FETI-DP-solver`, a highly scalable and robust implementation of FETI-DP in `PETSc` developed by O. Rheinbach as an important contribution of his doctoral thesis Rheinbach [110], see also Klawonn and Rheinbach [70].

Regarding the material modelling, the present work *exactly ties in with* Brands, Klawonn, Rheinbach, and Schröder [23] essentially carrying the simulations of arterial wall structures presented therein over to a massively parallel setting. This is a major leap forward, since the framework in Brands, Klawonn, Rheinbach, and Schröder [23] was limited by a semi-parallel, hence inherently non-scalable, software framework. Note that the DFG-project work continued at first with a focus on the interaction of the incompressibility constraint with FETI-DP, see Brinkhues, Klawonn, Rheinbach, and Schröder [25], also Brinkhues [24]. New results on efficient coarse spaces for incompressible and almost incompressible linear elasticity have been obtained, e.g., in Klawonn and Rheinbach [73] and Gippert [55] but have not been considered here. Instead, we used scalable coarse spaces for linear elasticity in three dimensions, again tying in with the setting used by Brands, Klawonn, Rheinbach, and Schröder [23].

For convenience, we present again the employed FETI-DP algorithms Algorithm C (Definition 1.3.6) and Algorithm D_E (Definition 1.3.7). These are defined in §1.3 where also the development of these algorithms is discussed.

Algorithm C [79]

The primal subspace $\widehat{W}_\Pi$ is induced by enforcing continuity of u on *all* the subdomain vertices $\mathcal{V}$ and the continuity of *all* edge averages $\overline{u}_{\mathcal{E}^{ik}}$ taken over all edges $\mathcal{E}$ in Γ.

Algorithm D_E [70]

The primal coarse space is generated by continuity of $\overline{u}_{\mathcal{E}^{ik}}$, i.e., by the continuity of the averages over all edges $\mathcal{E}$ in Γ.

Regarding the material models, we have decided to restrict our attention to the two stored energy densities Ψ_A and Ψ_B with optimally fitted parameter sets for the media and adventitia layers of the arterial wall structure. The two aforementionend energy densities are strongly anisotropic, quasi-incompressible and polyconvex, see, e.g., Balzani, Neff, Schröder, and Holzapfel [11]; see also Appendix A for an introduction to the anisotropic polyconvex framework and in particular to the implied existence theory.

More precisely, we consider the following two models for the media and adventitia layers in our simulations of soft-biological tissues:

Models for the Media and Adventitia Layers (cf. [23])

Model Ψ_A: (Balzani, Neff, Schröder, and Holzapfel [11])

$$\Psi_A\left(C, M^{(1)}, M^{(2)}\right) = \underbrace{c_1\left(\frac{I_1}{I_3^{1/3}} - 3\right)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\left(I_3^{\varepsilon_2} + \frac{1}{I_3^{\varepsilon_2}} - 2\right)}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2} \alpha_1 \left\langle K_3^{(i)} - 2\right\rangle_m^{\alpha_2}}_{\text{Fiber contributions}} .$$

Model Ψ_B: (Holzapfel, Gasser, and Ogden [64])

$$\Psi_B\left(C, M^{(1)}, M^{(2)}\right) = \underbrace{c_1\left(\frac{I_1}{I_3^{1/3}} - 3\right)}_{\text{Ground substance}} + \underbrace{\varepsilon_1\left(I_3^{\varepsilon_2} + \frac{1}{I_3^{\varepsilon_2}} - 2\right)^{\alpha_5}}_{\text{Volume penalty}} + \underbrace{\sum_{i=1}^{2} \frac{k_1}{2k_2}\left\{\exp\left(k_2\left\langle \frac{J_4^{(i)}}{I_3^{1/3}} - 1\right\rangle_m^2\right) - 1\right\}}_{\text{Fiber contributions}} .$$

It is intuitive that incompressibility is an essential constraint for soft biological tissues since these consist for the largest part of water. In Brinkhues, Klawonn, Rheinbach, and Schröder [25] the incompressibility constraint was treated using different Augmented Lagrange approaches which allows to satisfy the incompressibility constraint up to a prescribed tolerance. This is, however, outside of the scope of the present work. In `mparfeap`, we use a standard penalty approach for incompressibility as is easily inferred from the definition of the considered strain energies. For the discretization, we apply an $\overline{F}$-approach as in Brands, Klawonn, Rheinbach, and Schröder [23], Klawonn and Rheinbach [73], and Brinkhues, Klawonn, Rheinbach, and Schröder [25]. The particular technique is due to Simo [122] and reduces volume locking effects. For more in depth discussions, see, e.g., Appendix A; compare also the presentations in Klawonn and Rheinbach [73] and Brinkhues [24].

All presented computations are based on two parameter sets from Brands, Klawonn, Rheinbach, and Schröder [23] which have been fitted to experiments and an additional variant with relaxed volume penalty weight and exponent; see, e.g., Brinkhues, Klawonn, Rheinbach, and Schröder [25] for a related approach.

Material Parameters for the Adventitita and Media Layers (cf. [23])

Set	Model	Layer	c_1 [kPa]	ε_1 [kPa]	ε_2 [-]	α_1 [kPa]	α_2 [-]	α_3 [kPa]	α_4 [-]	α_5 [-]	k_1 [kPa]	k_2 [-]
2	Ψ_A	adv.	6.6	23.9	10.0	1503.0	6.3	–	–	–	–	–
		med.	17.5	499.8	2.4	30001.9	5.1	–	–	–	–	–
3	Ψ_B	adv.	6.2	101.0	10.0	–	–	–	–	3.0	6.0	20.0
		med.	10.7	207.1	9.7	–	–	–	–	10.0	1018.8	20.0
3 (relaxed)	Ψ_B	adv.	6.2	10.0	4.0	–	–	–	–	3.0	6.0	20.0
		med.	10.7	10.0	4.0	–	–	–	–	10.0	1018.8	20.0

The nonlinear boundary value problems underlying the simulations in the following sections have been formulated using the variational setting of anisotropic finite hyperelasticity. Essentially, the task at hand is to find a zero of a nonlinear residual functional which corresponds to an equilibrium of the interior and exterior virtual work.

Virtual Work

Interior Strain Energy:

$$\Pi^{\text{int}} : \mathcal{A} \to \mathbb{R}_0^+, \qquad \Pi^{\text{int}}(\varphi) = \int_\Omega \Psi(F, M^{(1)}, M^{(2)}) \ \text{dV} \ .$$

Interior Virtual Work (Internal Stresses):

$$G^{\text{int}} : T\mathcal{A} \to \mathbb{R}, \quad G^{\text{int}}|_\varphi(\chi) := \left. \frac{d}{d\varepsilon} \right|_{\varepsilon=0} \Pi^{\text{int}}(\varphi + \varepsilon\chi) \ .$$

External Virtual Work (Boundary Conditions):

$$\begin{aligned} G^{\text{ext}} : \quad & T\mathcal{A} \to \mathbb{R}, \\ G^{\text{ext}}|_\varphi(\chi) \quad := \quad & \underbrace{\int_{\partial\Omega_{\text{def}_N}} \langle t_N, \chi \circ \varphi^{-1} \rangle \ \text{dA}}_{\text{Pressure load}} \quad + \quad \underbrace{\int_\Omega \rho_0 \langle g, \chi \rangle \ \text{dV}}_{\text{Volume force}} \ . \end{aligned}$$

Note that this depicts a slight simplification since the $\overline{\text{F}}$-approach which requires a three-field formulation is not accounted for. We refer the reader to §A.9 and references therein for more details. The nonlinear equation for the residual $\mathcal{R}(u) = 0$ is first linearized then approximated by Tet-10, i.e., quadratic tetrahedral finite elements and solved using the Newton-Krylov-FETI-DP solver `mparfeap`. Note again that this corresponds to an equilibrium of interior and exterior virtual work.

Nonlinear Residual

$$\mathcal{R}(u) \ := \ G(u) \ := \ G^{\text{int}}(u) - G^{\text{ext}}(u) \ \stackrel{!}{=} \ 0 \ .$$

In simulations of finite continuum mechanics the discretized equilibrium equation $\mathcal{R}(u) = 0$ is in general impossible to solve without an effective homotopy method. More precisely, we have applied the adaptive failure-based load stepping strategy described in §1.4. This failure-based adaptive load stepping strategy *can* lead to complex sequences of failed and successful loading attempts. To clarify the presentation of the recorded convergence histories, we recall the following simple conventions from our introductory chapter: a launch of the Newton solver with a given load is called a **loading attempt** (see Definition 1.4.1). A **load step** is a successful loading attempt, i.e., a loading attempt for which the Newton solver signaled convergence (see Definition 1.4.2). In particular, the sequence of load steps is obtained from the sequence of all loading attempts by simply removing all failed loading attempts.

Further, for the sake of completeness we want to document the compiler optimization settings here.

Remark 3.1.1 (Compiler Settings)**.** *All `mparfeap` executables have been built with compiler optimizations turned on. For code compilation the `Intel Compiler Suite 11.1.069` was used on the Cray XT6m with the optimization flags `-O3 -ip -msse3`. All software components, i.e., `PETSc`, `FEAP-FW`, `libfw`, `libfddp`, and `mparfeap` have been compiled in this way.*

The previously available simulations of arterial wall stress states in the physiological regime obtained, e.g., in Brands, Klawonn, Rheinbach, and Schröder [23] and subsequent works were semi-parallel, hence inherently non-scalable. In the following sections we shall illustrate that this hurdle

was finally overcome with a new parallel scalable software framework presented in Chapter 2. Let us stress that the contributed library `libfw` was an important key to parallel scalability.

3.2 High-Resolution Computations of Arterial Wall Stresses – Ψ_A Set-2 – Algorithm C

We present some high-resolution simulations of wall stresses in atheriosclerotic arteries obtained for geometries reconstructed from IVUS (**I**ntra **V**ascular **U**ltra-**S**ound) patient specific measurement data. This leads to large-scale unstructured finite element problems for soft biological tissue models with loadings in the physiological regime imposed as an interior normal pressure of up to 500 [mmHg]. For this simulation, we have selected Ψ_A Set-2, i.e., the strain energy Ψ_A with the material parameter set 2 which was matched to experimental data in Brands, Klawonn, Rheinbach, and Schröder [23], see also our Table 1.3.

3.2.1 Physiological Arterial Wall Geometries

A hierarchy of arterial wall models was extracted from IVUS image data recorded at the "Westdeutsches Herzzentrum Essen" as part of the collaboration with Prof. Dr. med. R. Erbel and Dr. med. D. Böse in a joint DFG project, see [14]. For the present computations, we consider a simplified wall structure with three structural components: an isotropic plaque component, and two transversally isotropic incompressible soft tissue layers modelling media and adventitia.

Four levels of different resolutions with said structural components were generated by M. Lanser using the software package `Amira` [125].

Level	# D.o.f.	# Elements	z_{min}	z_{max}	Length [mm]
0	370 260	79 711	-0.025465	31.90331	31.928775
1	1 775 577	396 350	-0.038531	31.90550	31.944031
2	**13 429 950**	**3 170 800**	**-0.038531**	**31.89918**	**31.937711**
3	104 424 684	25 366 400	-0.038531	31.89918	31.937711

Although not perfectly identical, we shall consider these geometries for arterial segments with a length of approximately 3.2 [cm] as different resolution levels for the same computational domain. In turn, we obtain a scale of corresponding numerical finite element approximations to the respective boundary value problems. The resolution levels 0 and 1 have mainly been used for software testing purposes and to tune the loading strategy for our target resolution level 2. Level 3 is a refinement of level 2. It is too large for the Cray XT6m, i.e., the current implementation of our solver framework cannot solve this problem size on that machine yet. Even in the structured case which is a lot easier to handle, this problem size is near the limit for our current solver framework. In the unstructured case, the load balancing is more difficult and the coarse problem is usually larger. Thus, again, with the memory available on the Cray XT6m compute nodes this problem size is currently out of reach and we restrict our exposition to resolution level 2 with ca. 13 million unknows for the displacement vector u.

For the setup of the Dirichlet and Neumann boundary conditions, we followed Brands, Klawonn, Rheinbach, and Schröder [23]. The interior blood pressure is imposed as a Neumann boundary condition in the form of an interior normal pressure. This pressure was parametrized linearly in pseudo time. This is displayed in Table 3.1.

3.2.2 Solver Parameters and Tolerances

We have used the following `mparfeap` command string for these computations.

Pseudotime	Pressure [kPa]	[mmHg]
0	0	0
1	33.33	250
2	66.66	500

Table 3.1: High resolution wall-stress computations; Model artery 2; 13 million d.o.f.; Ψ_A Set-2; Linear parametrization for an internal pressure up to 500 [mmHg] at $t = 2$.

Command line options for `mparfeap`

```
aprun -n 256
 mparfeap-js-opt-mumps
-o=../artery_2:256:2013-10-24_09:53:37:1
-nw=512
-nwpp=2
-stamp=artery_2:256:2013-10-24_09:53:37:1
-load_stepper:strategy=adaptive(n=1600:initial=0.0:final=2.0:delay=3:alpha
    =1.3)
-load_stepper:use_linear_extrapolation=0
-newton:use_initial_guess_for_correction=1
-load_stepper:tecplot=1
-load_stepper:save_intermediate_solutions=1
-newton:atol=1e-5
-newton:rtol=2e-14
-feti:symmetric=1
-ksp_type gmres
-ksp_gmres_modifiedgramschmidt
-ksp_atol 1e-9
-ksp_rtol 2e-14
-ksp_gmres_restart 888
-ksp_max_it 888
-ksp_monitor_singular_value
-use_rho_local
-use_local_Kcc
-log_summary
```

Remark 3.2.1 (Tuning Options for Load Stepping)**.** *We see that the non-zero initial guess for λ was activated with*

```
-newton:use_initial_guess_for_correction=1
```

for this experiment and that linear extrapolation for u was turned off due to the option value

```
-load_stepper:use_linear_extrapolation=0
```

Note that this particular simulation failed to converge when the linear extrapolation feature was activated. A comparison illustrating the effect of the tuning options for the initial loading phase can be found in §3.5.

The load stepping strategy is the adaptive strategy previously described which is selected by

```
-load_stepper:strategy=adaptive(n=1600:initial=0.0:final=2.0:delay=3:alpha=1.3)
```

This sets the initial guess for the pseudotime increment to $2/1600 = 1/800$, the delay parameter to 3, the success multiplier for the load step size to $\alpha = 1.3$ and the failure multiplier $\beta = 0.5$ which is the default.

The FETI-DP coarse space built by `mparfeap` is generated from the input data precomputed by the `fddp` preprocessor application in a sequential step. To construct a strong scalability sequence with the coarse space for Algorithm C, we have run the `fddp` with the following command string:

Command line options for `fddp`

```
fddp -j=0
-i=Ias2
-nw=512
-nwpp=1:2:4:8
-d=../artery_2/corners+edges
-splitter=PARMETIS
-primal=ALL_NODES_ON_CORNERS:ALL_AVERAGES_ON_EDGES
```

This generates multiple input data sets for `mparfeap` with $1, 2, 4$ and 8 FETI-DP subdomains per MPI-instance while the actual graph partitioning computed by the `ParMetis` library Karypis, Schloegel, and Kumar [67] is only carried out once. For the present simulation, the model artery at resolution level 2 was decomposed into 512 FETI-DP subdomains. Due to the memory requirements for the coarse problem in Algorithm C, the simulation used only 12 of 24 cores per compute node. This doubles the available memory per MPI-process at the expense of a larger allocation of CPU cores.

3.2.3 FETI-DP Coarse Space

This simulation has been based on Algorithm C due to Klawonn, Widlund, and Dryja [79] in order to define the coarse space in Newton-Krylov-FETI-DP; see §1.3 for an introduction.

Algorithm C [79]

The primal subspace $\widehat{W}_\Pi$ is induced by enforcing continuity of u on *all* the subdomain vertices $\mathcal{V}$ and the continuity of *all* edge averages $\overline{u}_{\mathcal{E}^{ik}}$ taken over all edges $\mathcal{E}$ in Γ.

The edge averages required for Algorithm C have been implemented using a transformation of basis approach with the `FETI-DP-solver` due to Rheinbach [110], see also Klawonn and Rheinbach [70].

3.2.4 Convergence Statistics

On the following pages, we present the recorded convergence history for the computation and some statistical measurements in a series of plots.

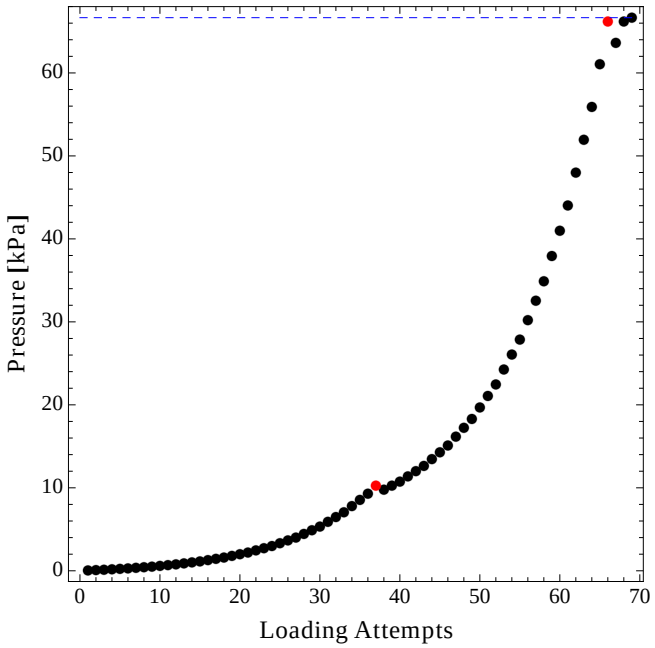


Figure 3.1: High resolution wall-stress computations; Model artery 2; Loading curve for interior normal pressure. Final state 500 [mmHg] $\approx$ 66.66 [kPa]; Load increments are adaptively computed; Two failed loading attempts are depicted in red.

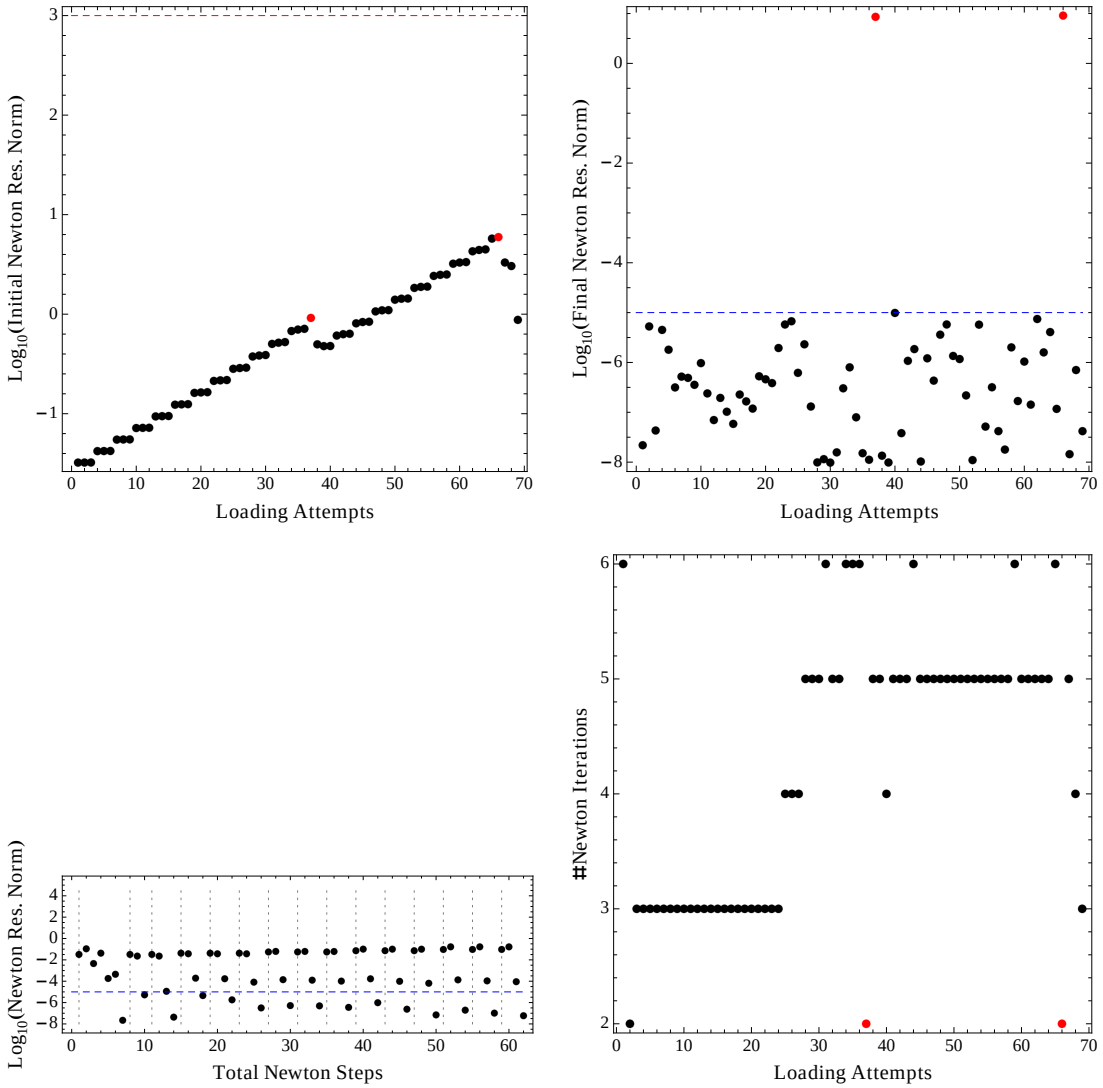


Figure 3.2: High resolution wall-stress computations; Numerical convergence statistics for the Newton scheme; Red dots mark failed attempts. Red stippled lines mark abortion criteria. Blue stippled lines mark convergence criteria. Residual norms for Newton's method show the first 15 load steps and show quadratic convergence (lower left).

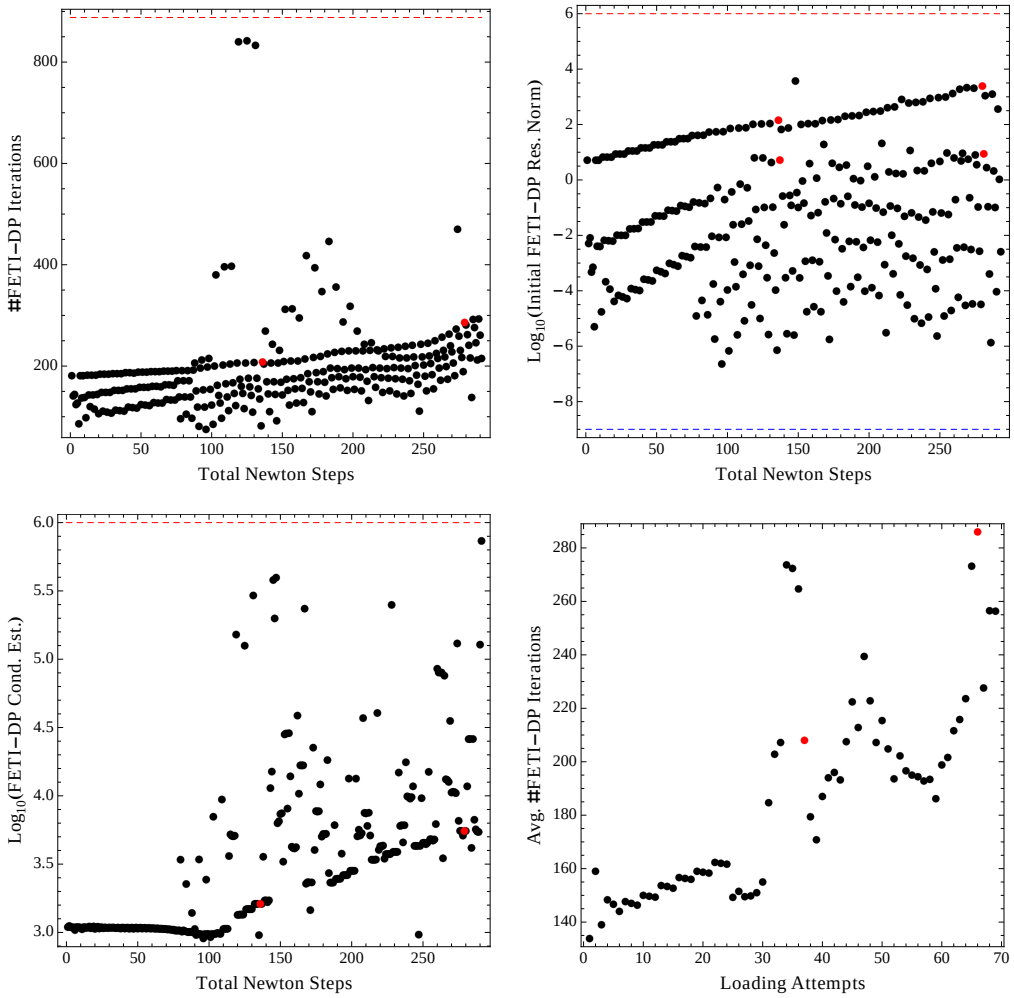


Figure 3.3: High resolution wall-stress computations; Numerical convergence statistics for FETI-DP; Red dots mark failed attempts. Red stippled lines mark abortion criteria. Blue stippled lines mark convergence criteria.

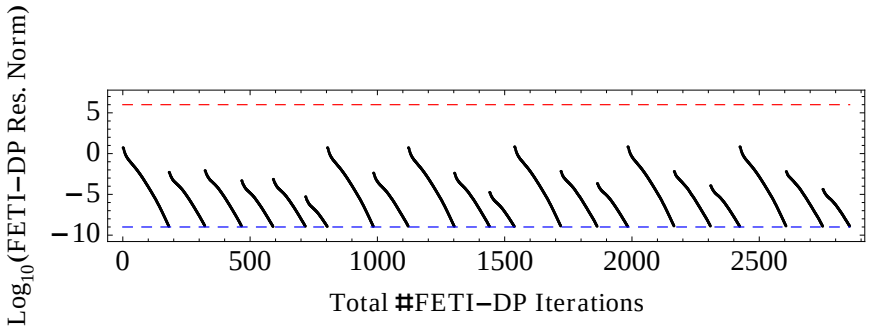


Figure 3.4: High-resolution wall-stress computations; History of the FETI-DP residual norms for the first six load steps; Red stippled lines mark abortion criteria. Blue stippled lines mark convergence criteria. *The initial FETI-DP residuals decrease during Newton iteration* due to the non-zero initial guess for FETI-DP given by $\lambda^{(n-1)}$.

3.2.5 Von Mises Equivalent Stresses

We have computed the von Mises equivalent stresses depicted in Figure 3.5, Figure 3.6 and Figure 3.7.

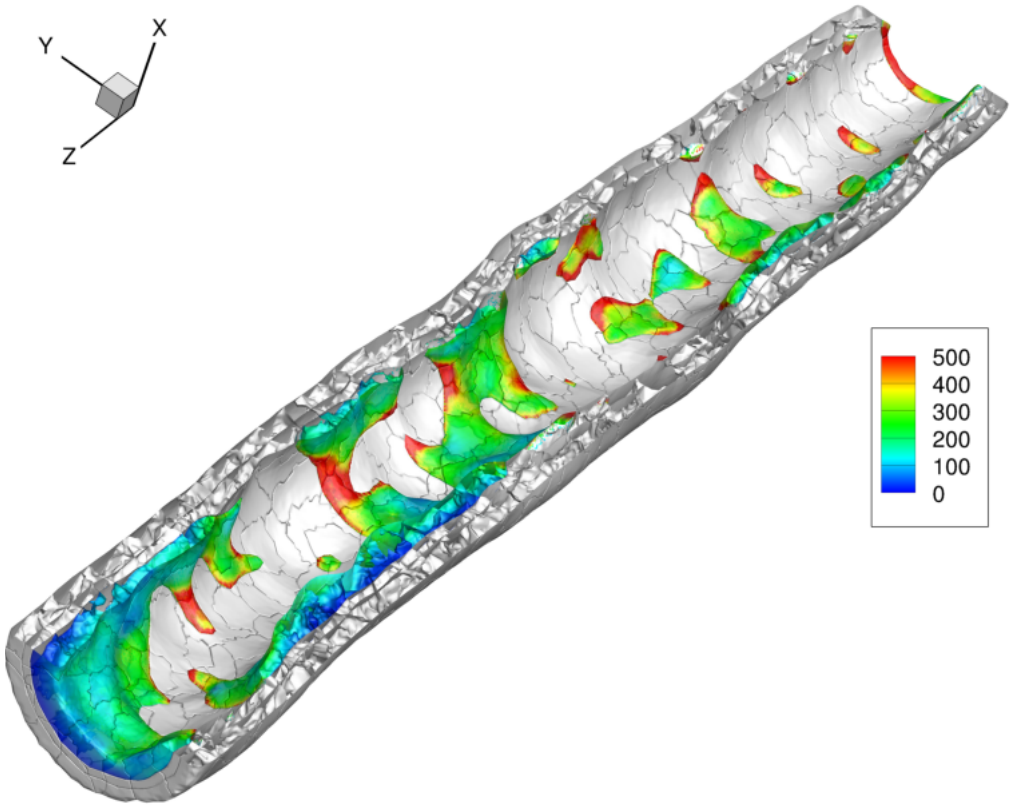


Figure 3.5: *Von Mises equivalent stresses displayed on the surface of the plaque component*; Model artery 2; 13 million d.o.f.; Ψ_A Set-2; Loadstep 56; Internal normal pressure $\approx$ 244 [mmHg]; 512 FETI-DP subdomains; 1024 Cray XT6m cores allocated; 12/24 cores per compute node to relax memory constraints; FETI-DP coarse space Algorithm C [79]; Finite element assembly in `FEAP -JS` via `libfw-js`, see e.g., [20, 23]; Embedded `FETI-DP-solver` [70, 110] based on `PETSc`; Activated non-zero initial guess for the Lagrange multipliers λ in GMRES; Deactivated linear extrapolation for the displacement u.

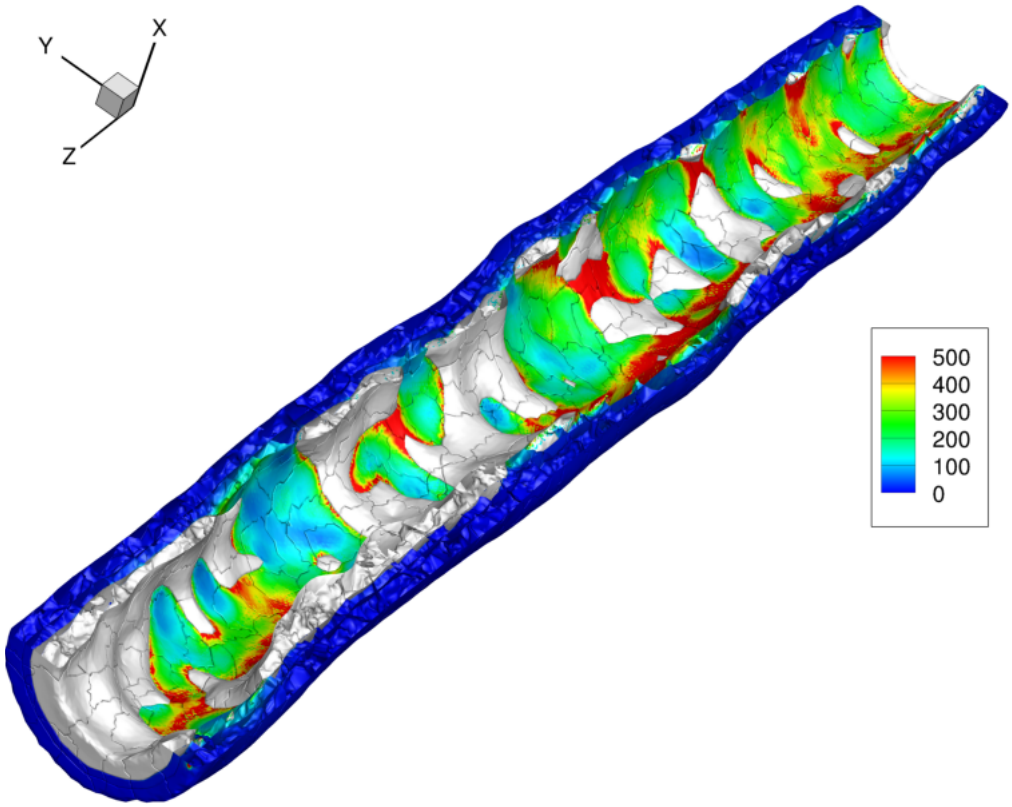


Figure 3.6: *Von Mises equivalent stresses displayed on the healthy sections of the interior surface of the media layer*; Model artery 2; 13 million d.o.f.; Ψ_A Set-2; Loadstep 56; Internal normal pressure ≈ 244 [mmHg]. Note the stress localizations on the surface of the adventitia layer between closely neighbouring plaque components and near the boundaries of plaque components. 512 FETI-DP subdomains; 1024 Cray XT6m cores allocated; 12/24 cores per compute node to relax memory constraints; FETI-DP coarse space Algorithm C [79]; Finite element assembly in `FEAP-JS` via `libfw-js`, see e.g., [20, 23]; Embedded `FETI-DP-solver` [70, 110] based on `PETSc`; Activated non-zero initial guess for the Lagrange multipliers λ in GMRES; Deactivated linear extrapolation for the displacement u.

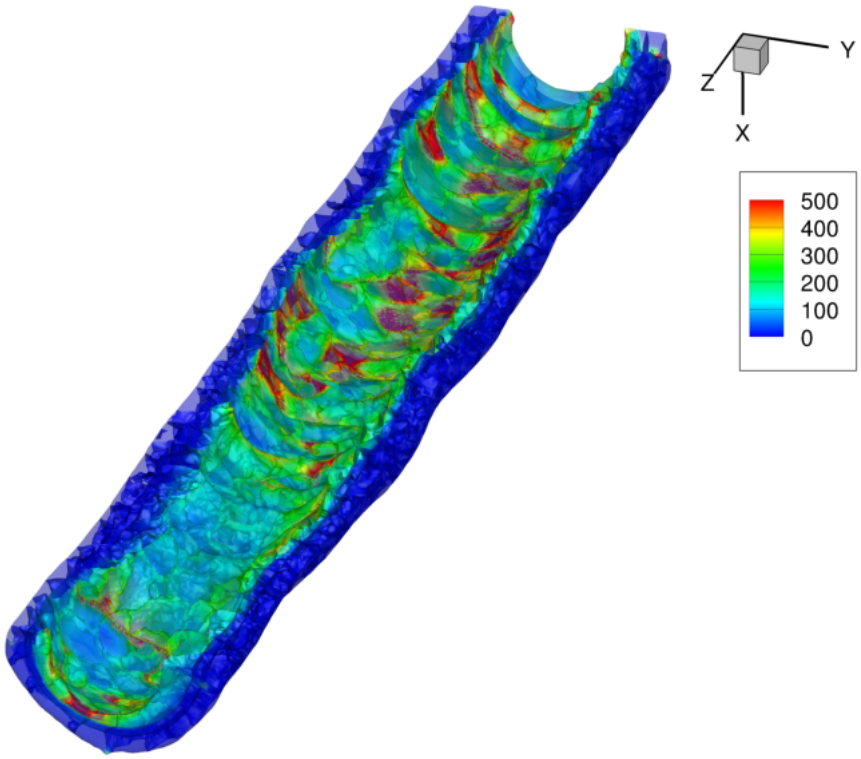


Figure 3.7: *Von Mises equivalent stresses displayed transparently for all material components*; Model artery 2; 13 million d.o.f.; Ψ_A Set-2; Loadstep 56; Internal normal pressure ≈ 244 [mmHg]. 512 FETI-DP subdomains; 1024 Cray XT6m cores allocated; 12/24 cores per compute node to relax memory constraints; FETI-DP coarse space Algorithm C [79]; Finite element assembly in `FEAP-JS` via `libfw-js`, see e.g., [20, 23]; Embedded `FETI-DP-solver` [70, 110] based on `PETSc`; Activated non-zero initial guess for the Lagrange multipliers λ in GMRES; Deactivated linear extrapolation for the displacement u.

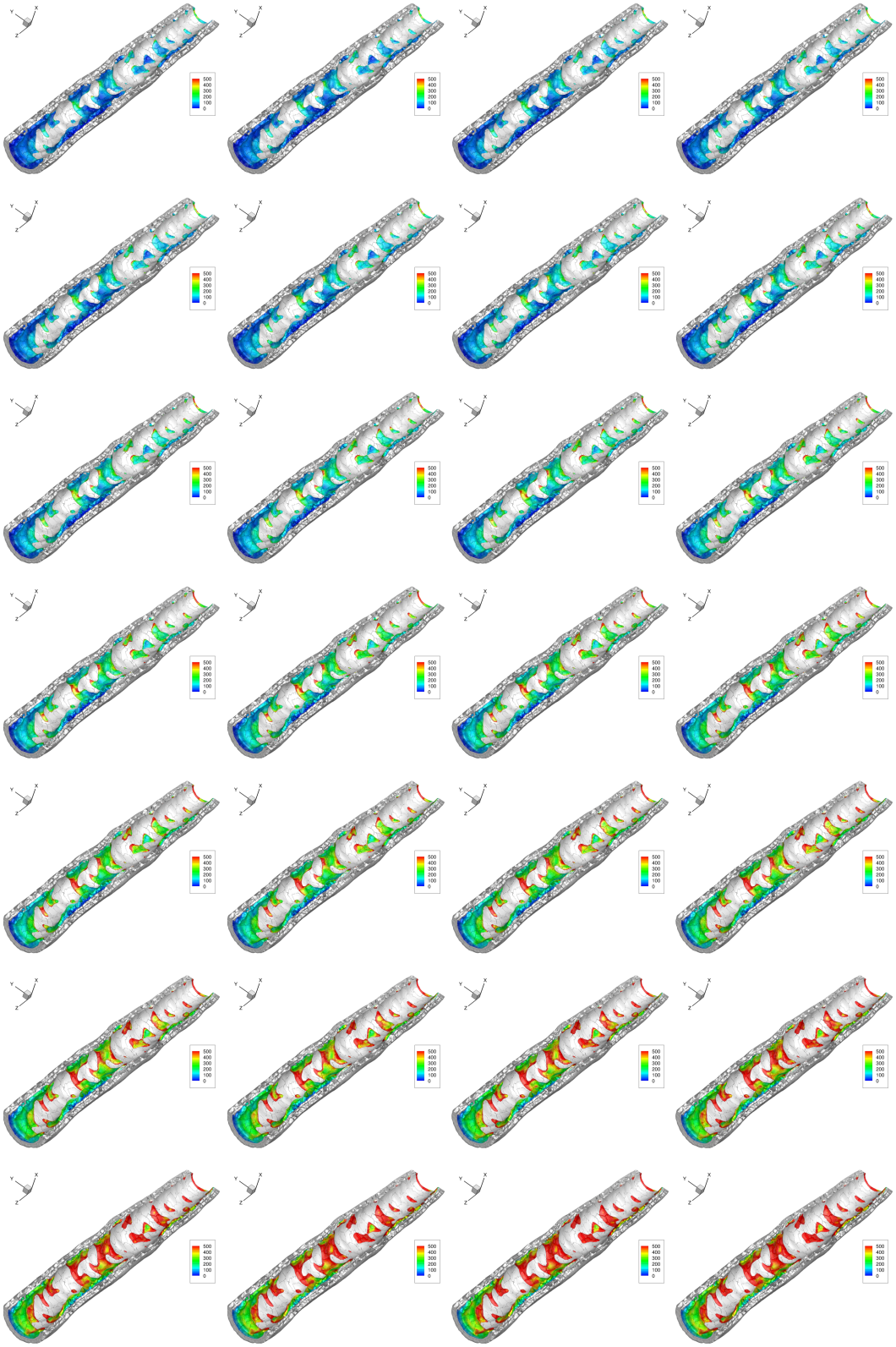


Figure 3.8: Model artery 2; 13 million d.o.f.; Ψ_A Set-2; Von Mises equivalent stresses on the surface of the plaque component during loadsteps 40 to 67. This corresponds to an internal normal pressure increasing from 10 [mmHg] to 500 [mmHg].

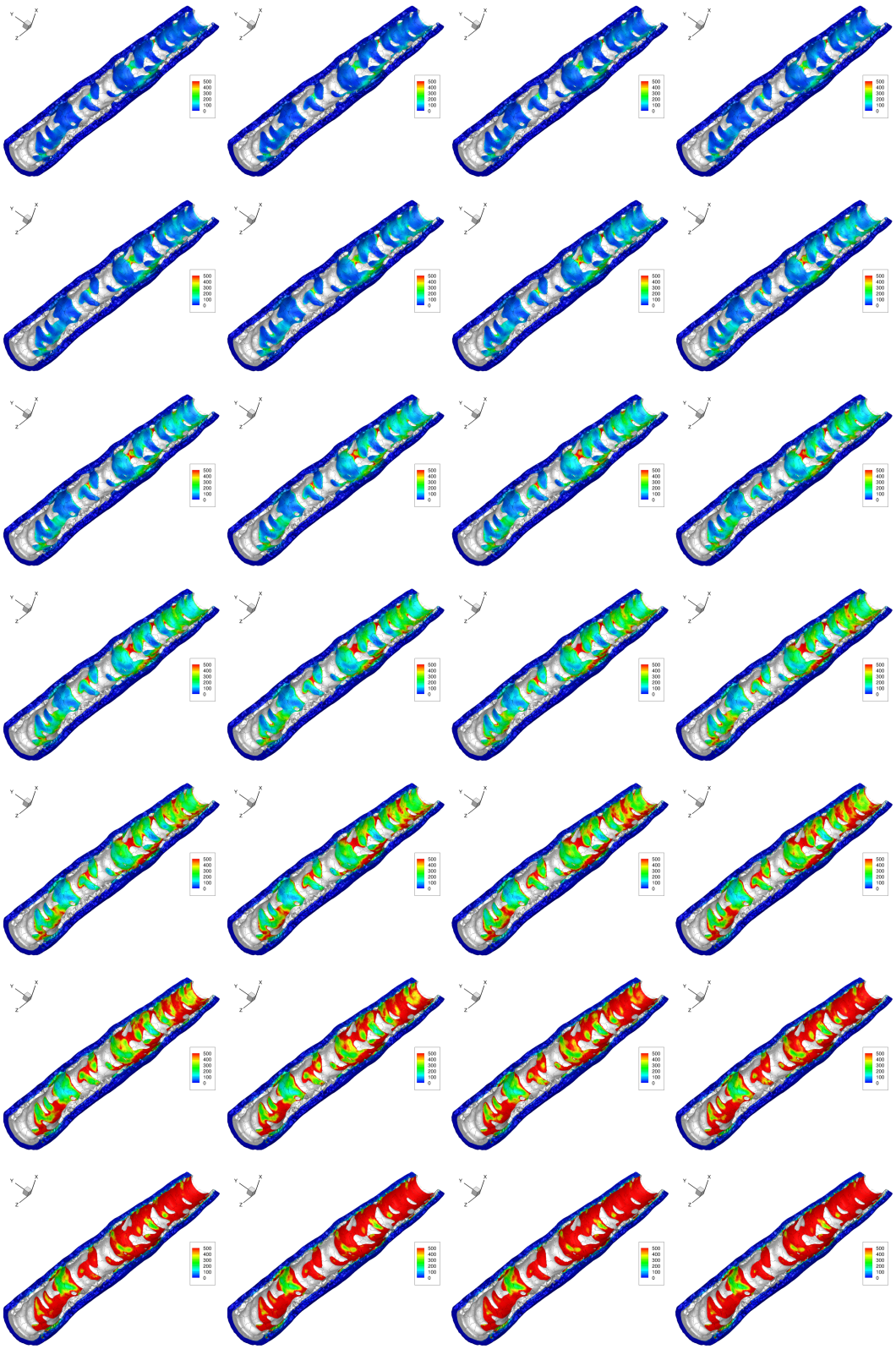


Figure 3.9: Model artery 2; 13 million d.o.f.; Ψ_A Set-2; Von Mises equivalent stresses on the internal surface of the media component; Loadsteps 40 to 67. This corresponds to an internal normal pressure increasing from 10 [mmHg] to 500 [mmHg].

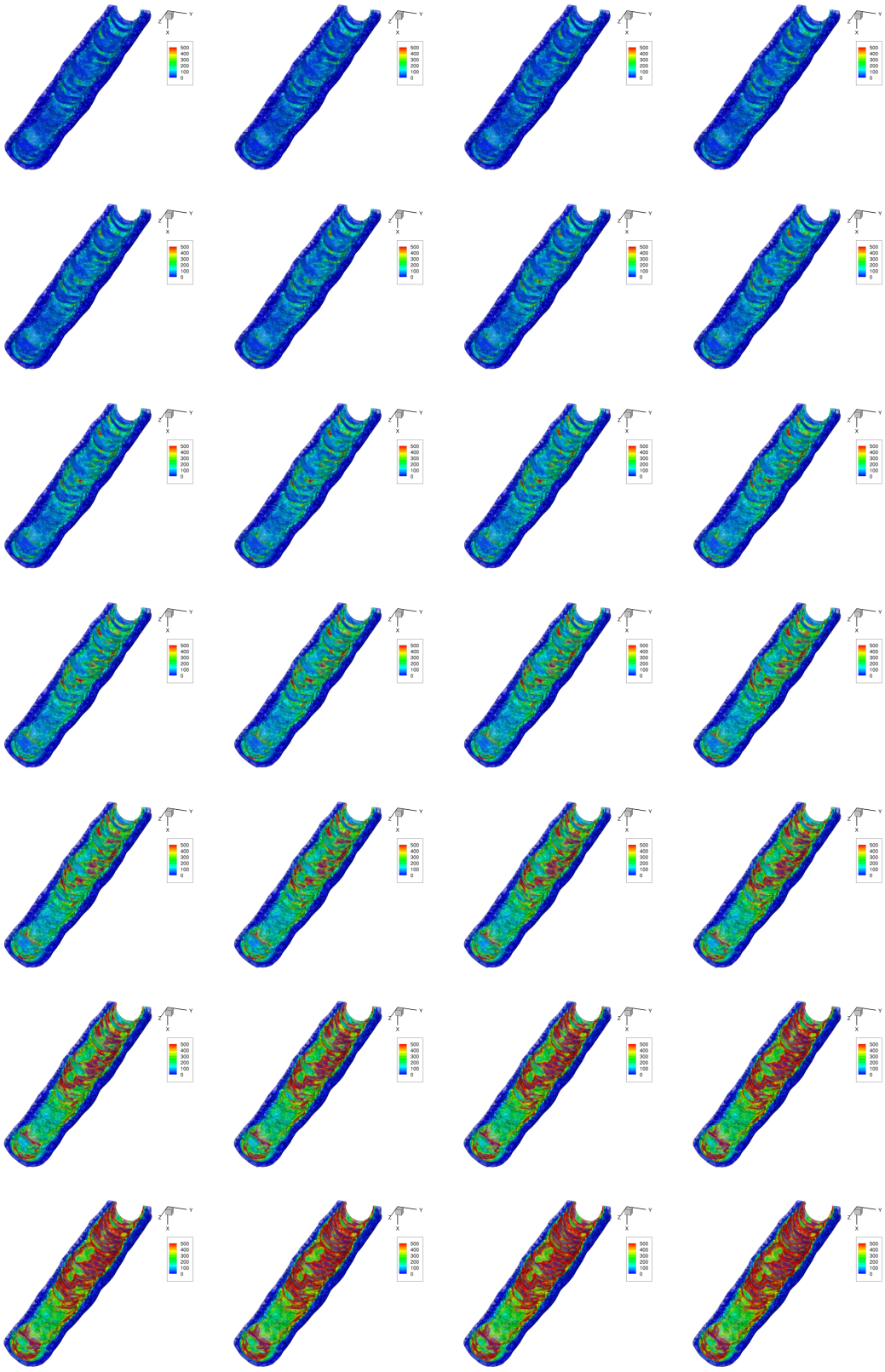


Figure 3.10: Model artery 2; 13 million d.o.f.; Ψ_A Set-2; Von Mises equivalent stresses displayed transparently for all material components; Loadsteps 40 to 67; Internal pressure $10 - 500$ [mmHg].

3.3 Strong Scalability – Model Artery 2 – Ψ_A Set-2 – Algorithm C

We now present strong scalability results for the Newton-Krylov-FETI-DP solver `mparfeap` obtained for the model artery at resolution level 2. Strong parallel scalability is a major strength of the `FETI-DP-solver` and this translates perfectly to the nonlinear setting considered here. To study strong scalability of the Newton-Krylov-FETI-DP solver `mparfeap` model artery 2 was decomposed into 512 subdomains. These subdomains were allocated to an increasing number of processor cores: 128, 256 and 512. The ideal speedups are a factor of 2 for each scaling level and corresponding efficiencies are presented in Table 3.3. For the generation of the input data sets from a `FEAP` problem description, we have used the `fddp` preprocessor application.

As in the previous section, we consider again the identical problem setup for model artery 2 with material model Ψ_A Set-2. The final blood pressure is reduced to 250 [mmHg] in order to reduce the total computational time.

Pseudotime	**Pressure**	
	[kPa]	**[mmHg]**
0	0	0
1	33.33	250

Table 3.2: Strong scalability; Model artery resolution level 2; Material Ψ_A with parameter set 2; Linear parametrization of the interior pressure loading up to 250 [mmHg] at $t = 1$.

Command line options for `mparfeap`

```
aprun -n 512
mparfeap-js-opt-mumps
-o=../artery_2:512:2013-10-27_10:23:54:1
-nw=512
-nwpp=1
-stamp=artery_2:512:2013-10-27_10:23:54:1
-log_summary
-load_stepper:strategy=adaptive(n=800:initial=0.0:final=1.0:delay=3:alpha=1.3)
-load_stepper:use_linear_extrapolation=0
-newton:use_initial_guess_for_correction=1
-newton:clear_initial_guess_for_correction_on_reset=0
-load_stepper:tecplot=0
-load_stepper:save_intermediate_solutions=0
-newton:atol=1e-5
-newton:rtol=2e-14
-feti:symmetric=1
-ksp_type gmres
-ksp_gmres_modifiedgramschmidt
-ksp_atol 1e-9
-ksp_rtol 2e-14
-ksp_gmres_restart 888
-ksp_max_it 888
-ksp_monitor_singular_valuees
-use_rho_local
-use_local_Kcc
```

For this simulation we have reused the input data from the simulation presented in §3.2. However, we note that the simulation is terminated at final pseudotime $t = 1$ for this strong scalability experiment. This corresponds to a maximal loading of 250 [mmHg].

Cores	Subdomains	Problem Size [# d.o.f.]	Time [h:m:s]	[s]	Efficiency
128	512	13 429 950	51:38:59	185 939	**100.00%**
256	512	13 429 950	24:49:40	89 380	**104.01%**
512	512	13 429 950	12:59:02	46 742	**95.61%**

Table 3.3: Strong scalability; Model artery 2; 13 million d.o.f.; Ψ_A Set-2; 12/24 cores per compute node to relax memory constraints; 512 FETI-DP subdomains; FETI-DP coarse space Algorithm C [79]; Embedded `FETI-DP-solver` [70, 110] based on `PETSc`; Activated non-zero initial guess for the Lagrange multipliers λ in GMRES; Deactivated linear extrapolation for the displacement u.

Remark 3.3.1 (Tuning Options for Load Stepping)**.** *The non-zero initial guess for λ was activated for this experiment. The linear extrapolation for u was turned off. Note that the simulation failed to converge when the linear extrapolation feature was activated. A comparison illustrating the effect of the tuning options for the initial loading phase can be found in §3.5.*

3.3.1 FETI-DP Coarse Space

This simulation was based on Algorithm C due to Klawonn, Widlund, and Dryja [79] in order to define the coarse space in Newton-Krylov-FETI-DP; see §1.3 for an introduction.

Algorithm C [79]

The primal subspace $\widehat{W}_\Pi$ is induced by enforcing continuity of u on *all* the subdomain vertices $\mathcal{V}$ and the continuity of *all* edge averages $\overline{u}_{\mathcal{E}^{ik}}$ taken over all edges $\mathcal{E}$ in Γ.

The edge averages required for Algorithm C have been implemented using a transformation of basis approach with the `FETI-DP-solver` due to Rheinbach [110] and Klawonn and Rheinbach [70].

For the convergence history of the computations, see section §3.2 which is identical up to the maximal interior normal pressure of 250 [mmHg] here prescribed.

3.3.2 Tabulated Results

The results collected in Table 3.3 show the reduction of the total application runtime and the corresponding efficiency ratios for an increasing number of CPU cores while the problem size has been kept fixed. An efficiency of 100% is perfect in a theoretical setting. Ratios of more than 100% may seem strange, but they are quite common in practice and caused by variations in disk I/O, network as well as CPU cache effects.

3.4 Weak Scalability – Ψ_B Set-3 Relaxed – Algorithm D_E

In this section, we report results on the weak nonlinear scalability of Newton-Krylov-FETI-DP. To this end a series of simple tension tests is investigated. For the material model, the polyconvex soft-tissue material model Ψ_B with parameters from Set-3 Brands, Klawonn, Rheinbach, and Schröder [23] was used. The simulations were run on the Cray XT6m supercomputer at Universität Duisburg-Essen and discrete finite element systems with up to 134 million degrees of freedom were solved on up to 4096 of a total of 4122 CPU cores. All of the weak scalability experiments showed a peak memory usage of 80% and above measured on the root compute node. The best results were obtained close to complete saturation of the available random access memory of the fully allocated machine.

In the following subsections, we present a gallery of four experiments based on simple tension tests. These are based on the following associated computational domains and structured non-overlapping domain decomposition sequences thereof:

1. Cuboid (§3.4.4) – The bottom half of a unit cube, $n = 2, 3, 4$:

$$\begin{aligned} H^{(n)}/h^{(n)} &= 13\ , \\ H^{(n)} &= \tfrac{1}{2^n}\ , \\ \Omega^{(n)} &= [0,1]^2 \times [0,\tfrac{1}{2}]\ . \end{aligned}$$

2. Plate $2H$ (§3.4.5) – A sequence of plates of height $2H$, $n = 3, 4, 5, 6$:

$$\begin{aligned} H^{(n)}/h^{(n)} &= 11\ , \\ H^{(n)} &= \tfrac{1}{64}\ , \\ \Omega^{(n)} &= [0,2^n H] \times [0,2^{(n-1)} H] \times [0,2H]\ . \end{aligned}$$

3. Plate $4H$ (§3.4.6) – A sequence of plates of height $4H$, $n = 2, 3, 4, 5$:

$$\begin{aligned} H^{(n)}/h^{(n)} &= 13\ , \\ H^{(n)} &= \tfrac{1}{32}\ , \\ \Omega^{(n)} &= [0,2^n H] \times [0,2^{(n-1)} H] \times [0,4H]\ . \end{aligned}$$

4. Cube (§3.4.7) – A unit cube, $n = 1, 2, 3, 4$:

$$\begin{aligned} H^{(n)}/h^{(n)} &= 11\ , \\ H^{(n)} &= \tfrac{1}{2^n}\ , \\ \Omega^{(n)} &= [0,1]^3\ . \end{aligned}$$

The corresponding boundary value problems are tension tests for different geometry sequences and they were set up to be as similar as possible. For FETI-DP methods, it is natural to parametrize scaling sequences in terms of the H/h quotient. The typical known condition number bounds for model problems, such as linear elasticity, are formulated in terms of H/h. A larger value of the H/h ratio corresponds to a larger local problem size per subdomain, i.e., more degrees of freedom per substructure. Note that this will usually improve parallel scalability in FETI-DP, since the perfectly scalable local solution phases of the FETI-DP method gain more weight in the simulation.

We report only the best results for each experiment. In all cases this corresponds to the maximal value of H/h for which the given experiment still fitted into the available memory of the Cray XT6m, i.e., into 32 GB of shared memory per compute node.

Remark 3.4.1 (Tuning Options for Load Stepping)**.** *Both load stepping tuning options which are available in* `mparfeap` *were activated, i.e., linear extrapolation for the displacement* u *and the*

dual initial guess for λ in FETI-DP for the weak scalability experiments to be presented in this section. In fact, we could not have hoped to obtain the presented results in a reasonable amount of time without application of these techniques, as is illustrated in §3.5.

3.4.1 FETI-DP Coarse Space

Due to memory limitations, we consider the FETI-DP coarse space defined by Algorithm D_E here. This algorithm was introduced in Klawonn and Rheinbach [70] and is more conservative than Algorithm C, since the vertices are removed from the primal space.

Algorithm D_E [70]

The primal coarse space is generated by continuity of $\overline{u}_{\mathcal{E}^{ik}}$, i.e., by the continuity of the averages over all edges $\mathcal{E}$ in Γ.

The edge averages required for Algorithm D_E have been implemented using a transformation of basis approach with the `FETI-DP-solver` due to Rheinbach [110] and Klawonn and Rheinbach [70].

3.4.2 Setup of Tension Tests

We consider geometry sequences obtained by partitioning cuboidal domains in a structured way, i.e., they are decomposed into cubic subdomains for FETI-DP. Les us introduce the boundary conditions. Each tension test is realized by setting Dirichlet boundary conditions on two opposing faces of the cuboidal computational domain $\Omega^{(n)}$, $n \in \mathbb{N}$, of the scaling sequence. It is fixed in the $x = 0$-plane by prescribing a zero displacement $u = 0$. On the opposite parallel face, a non-zero displacement corresponding to 4% length change (relative to the total length of the global domain in x-direction) is imposed and points in the positive x-direction.

All 4 scaling experiments: the Cuboid, the Plate $2H$, the Plate $4H$ and the Cube were thus set up in an as comparable as possible way to render them at least remotely comparable. The problem sizes are, however, different for each weak scalability experiment.

For the following weak scalability experiments we have used the material strain energy Ψ_B with parameter Set-3 for the adventitia layer. and relaxed the volume constraint. I.e., we have set the parameters for the incompressibility penalty to $\varepsilon_1 = 10$ and $\varepsilon_2 = 4$ while leaving all other material parameters from Set-3 unchanged, see Table 1.3. This saves some Newton iterations in the first load steps in comparison to the same problem with Ψ_A Set-2, hence saving precious compute time on the fully allocated Cray XT6m supercomputer. For all weak scalability experiments, we have used the two preferred fiber directions

$$a_1 = \begin{bmatrix} 0.656 \\ -0.755 \\ 0.0 \end{bmatrix}, \quad \text{and} \quad a_2 = \begin{bmatrix} 0.656 \\ 0.755 \\ 0.0 \end{bmatrix}.$$

Both fiber directions lie in the $z = 0$-plane and enclose an angle of approximately 98°, which has been used previously in Brands, Klawonn, Rheinbach, and Schröder [23] for the enclosed fiber angle in the adventitia layer.

The main focus of our presentation are new results on numerical and parallel weak scalability for Newton-Krylov-FETI-DP for the described tension tests. However, to put our nonlinear results into perspective, we prepend a reference benchmark problem using linear elasticity.

3.4.3 Cuboid – Linear Elasticity – Algorithm D_E

In order to illustrate potential numerical and parallel scalability on the Cray XT6m to gauge expectations for the considered nonlinear problems, we first present some results for linear elasticity obtained using our solver `mparfeap`, i.e., the software environment described in Chapter 2. First, we present the problem described in §3.4.4 using linear elasticity for the material model. The computational domain is a cuboid $\Omega = [0,1] \times [0,1] \times [0,1/2]$ which is decomposed into cubic subdomains for the FETI-DP method.

The Cuboid tension test is formulated using the following scaling sequence for $n = 2, 3, 4$:

$$\begin{aligned} H^{(n)}/h^{(n)} &= 13\,, \\ H^{(n)} &= \tfrac{1}{2^n}\,, \\ \Omega^{(n)} &= [0,1]^2 \times [0, \tfrac{1}{2}]\,. \end{aligned}$$

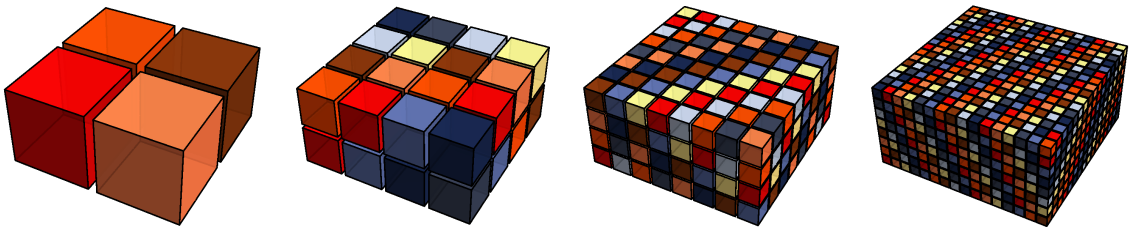


Figure 3.11: Weak scalability; Cuboid; $H/h = 13$; Sequence of partitioned meshes for the Cuboid geometry $\Omega := [0,1]^2 \times [0, \frac{1}{2}]$.

Cuboid – Solver Parameters and Tolerances

Command line options for `mparfeap`

```
aprun -n 2048
mparfeap-js-opt-mumps
-o=edges_Hh13_linear_elasticity_half_nodes:2048:2012-10-09_19:29:25:1
-nw=2048
-nwpp=1
-stamp=edges_Hh13_linear_elasticity_half_nodes:2048:2012-10-09_19:29:25:1
-log_summary
-load_stepper:strategy=linear(n=1:initial=0.0:final=4.0)
-load_stepper:use_linear_extrapolation=1
-newton:use_initial_guess_for_correction=1
-load_stepper:tecplot=0
-load_stepper:save_intermediate_solutions=0
-load_stepper:use_pseudo_time_for_dirichlet_boundary_interpolation=1
-newton:atol=1e-7
-newton:rtol=2e-14
-feti:symmetric=1
-ksp_type gmres
-ksp_gmres_modifiedgramschmidt
-ksp_rtol 1e-7
-ksp_atol 1e-40
-ksp_gmres_restart 888
```

```
-ksp_max_it 888
-ksp_monitor_singular_value
-use_rho_local
-use_local_Kcc
```

Cuboid – Tabulated Results

Cores	Global Problem [# d.o.f]	Local Problem [# d.o.f]	Coarse Problem [# d.o.f]	Dual Problem [# d.o.f]
2048	109 028 403	59 049	19 572	11 041 113
256	13 759 515	59 049	2 496	1 262 949
32	1 752 975	59 049	294	128 091

Table 3.4: Weak scalability; Cuboid; $H/h = 13$; *Linear elasticity*; Problem sizes.

Cores	FETI-DP Cond.*	Eff. [%]	FETI-DP It.	Eff. [%]	Time [s]	Eff. [%]
32	11.63	100.0	26	100.0	138.7s	100.0
256	12.70	91.6	28	92.9	153.5s	90.4
2048	13.00	89.5	28	92.9	180.8s	76.7

Table 3.5: Weak scalability; Cuboid; $H/h = 13$; *Linear elasticity*; Numerical and parallel scalability; Tolerances $\text{atol}_{\text{Newton}} = 10^{-7}$ and $\text{rtol}_{\text{FETI-DP}} = 10^{-7}$; 12/24 CPU cores per compute node; Cray XT6m; FETI-DP coarse space Algorithm D_E [70, 110].

From Table 3.5 we see that a satisfactory efficiency of 76.7% is achieved. Note that the Cray XT6m machine is not equipped with the most recent version of the Cray interconnect but still uses the older SeaStar2+. The coarse space from Algorithm D_E was constructed for compressible linear elastity. This is reflected in the numerical experiment and we see only a minor increase of the FETI-DP iterations and a corresponding efficiency of 92.9%.

3.4.4 Cuboid – Ψ_B Set-3 Relaxed – Algorithm D_E

The Cuboid tension test is formulated using the following scaling sequence for $n = 2, 3, 4$:

$$\begin{aligned} H^{(n)}/h^{(n)} &= 13\,, \\ H^{(n)} &= \tfrac{1}{2^n}\,, \\ \Omega^{(n)} &= [0,1]^2 \times [0, \tfrac{1}{2}]\,. \end{aligned}$$

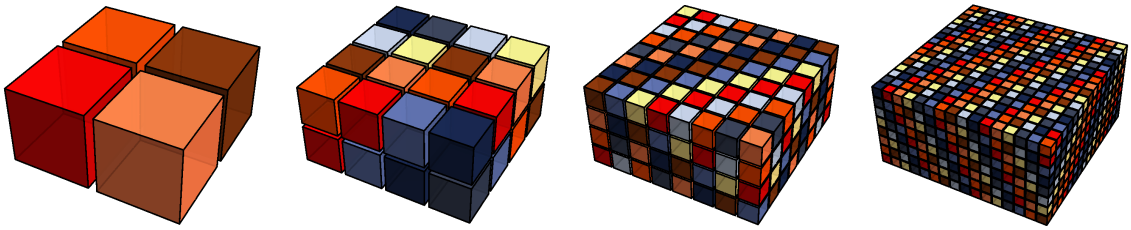


Figure 3.12: Weak scalability; Cuboid; $H/h = 13$; Sequence of partitioned meshes for the cuboid geometry $[0,1]^2 \times [0, \frac{1}{2}]$.

Cuboid – Solver Parameters and Tolerances

We have used equivalent parameters for all problems in the weak scalability sequence. For the largest problem in the scaling sequence using 2048 cores, we have used the following command string for `mparfeap`:

Command line options for `mparfeap`

```
aprun -n 2048
mparfeap-js-opt-mumps
-o=../edges_Hh13_psi_b_relaxed_half_nodes:2048:2012-10-10_09:00:19:1
-nw=2048
-nwpp=1
-stamp=edges_Hh13_psi_b_relaxed_half_nodes:2048:2012-10-10_09:00:19:1
-load_stepper:strategy=adaptive(n=512:initial=0.0:final=4.0:alpha=1.5:delay=2)
-load_stepper:use_linear_extrapolation=1
-newton:use_initial_guess_for_correction=1
-load_stepper:tecplot=0
-load_stepper:save_intermediate_solutions=0
-load_stepper:use_pseudo_time_for_dirichlet_boundary_interpolation=1
-newton:atol=1e-7
-newton:rtol=2e-14
-feti:symmetric=1
-ksp_type gmres
-ksp_gmres_modifiedgramschmidt
-ksp_rtol 2e-14
-ksp_atol 1e-9
-ksp_gmres_restart 444
-ksp_max_it 888
-ksp_monitor_singular_value
-use_rho_local
-use_local_Kcc
```

Cores	Global Problem [# d.o.f]	Local Problem [# d.o.f]	Coarse Problem [# d.o.f]	Dual Problem [# d.o.f]
2048	109 028 403	59 049	19 572	11 041 113
256	13 759 515	59 049	2 496	1 262 949
32	1 752 975	59 049	294	128 091

Table 3.6: Weak scalability; Cuboid; $H/h = 13$; Problem sizes.

Cuboid – Tabulated Results

Cores	Loading Attempts	Eff. [%]	Newton It. (Σ)	Eff. [%]	FETI-DP It. (Σ)	Eff. [%]	FETI-DP It. ($\oslash$)	Eff. [%]	FETI-DP Cond.* ($\oslash$)	Eff. [%]
32	24	100	78	100	1 552	100	19.90	100	21.99	**100**
256	24	100	90	87	1 935	80	21.50	93	26.37	**83**
2048	24	100	97	80	2 389	65	24.63	81	34.31	**64**

Cores	FETI-DP ($\boldsymbol{\Sigma}$) [s]	Eff. [%]	FETI-DP ($\oslash$) [s]	Eff. [%]	GMRES ($\boldsymbol{\Sigma}$) [s]	Eff. [%]	GMRES ($\oslash$) [s]	Eff. [%]	Run [s]	Eff. [%]
32	9 380	100	120.25	100	2 148	100	27.54	100	11 095.6	**100**
256	12 189	77	135.43	89	3 052	70	33.91	81	14 216.9	**78**
2048	16 453	57	169.62	71	5 264	41	54.27	51	18 825.1	**59**

Table 3.7: Weak scalability; Cuboid; $H/h = 13$; Timings and efficiences for parallel scalability; FETI-DP coarse space Algorithm D_E [70, 110]. The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

Cuboid – Newton Iterations

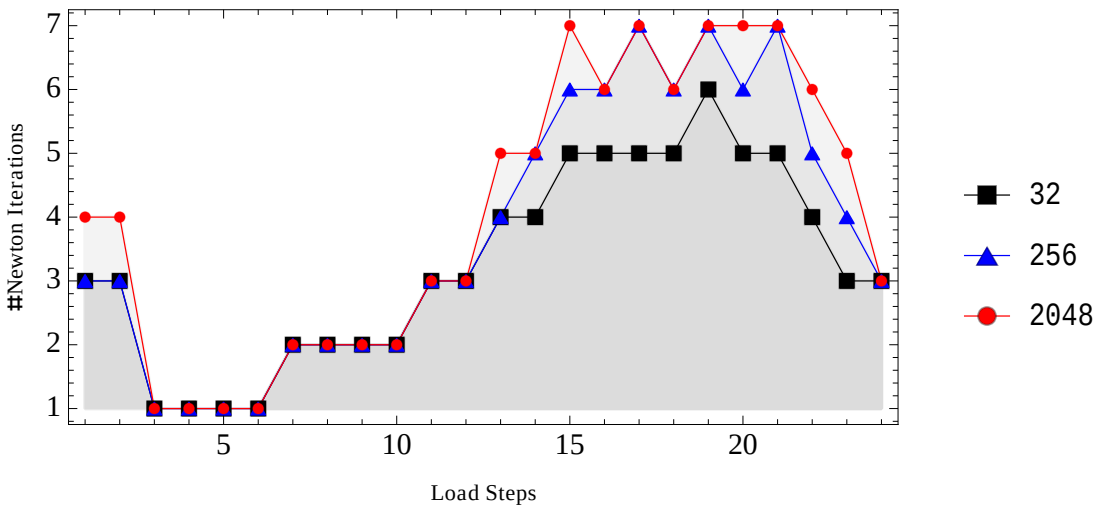


Figure 3.13: Weak scalability; Cuboid; $H/h = 13$; Newton iterations; The number of Newton iterations increases with the number of CPU cores. Thus the total number of FETI-DP systems to be solved increases and parallel scalability is adversely affected; see also §3.4.8.

Cuboid – Numerical Scalability and Convergence Statistics

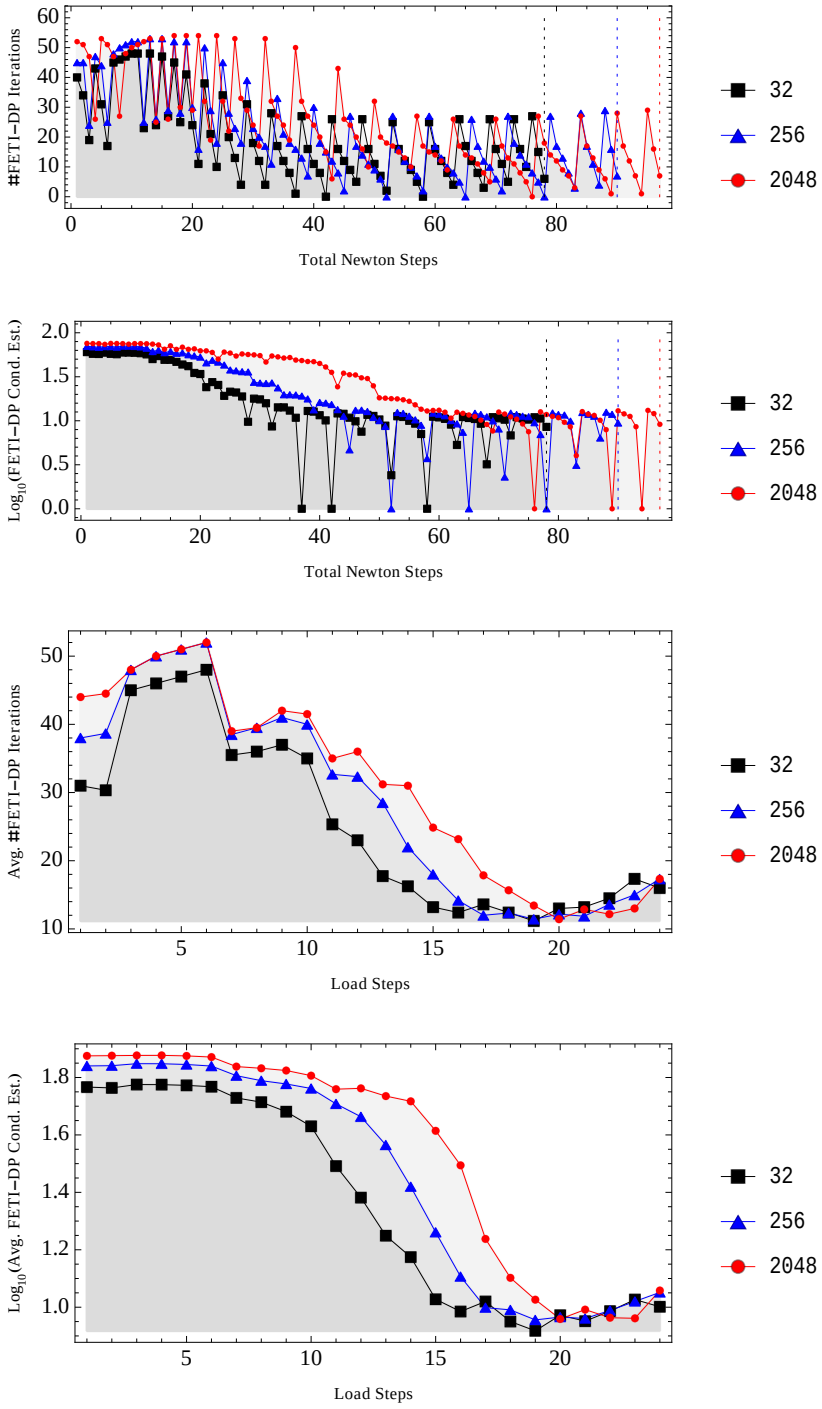


Figure 3.14: Weak scalability; Cuboid; $H/h = 13$; FETI-DP iterations and condition number estimates; FETI-DP coarse space Algorithm D_E [70, 110]. Note the increasing number of the total number of Newton iterations; The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

Cuboid – Parallel Scalability and Timing Statistics

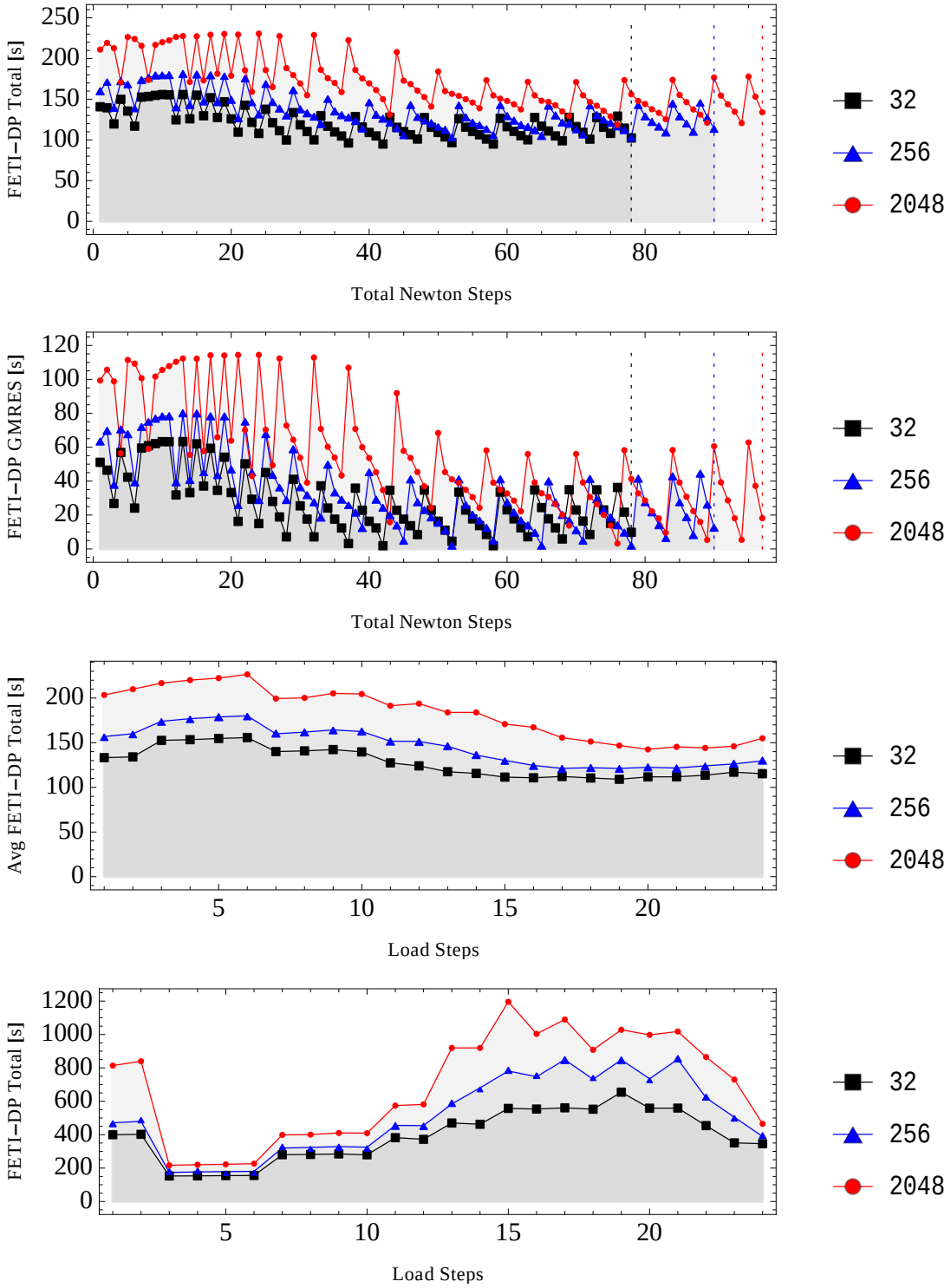


Figure 3.15: Weak scalability; Cuboid; $H/h = 13$; Newton-Krylov-FETI-DP timings; FETI-DP coarse space Algorithm D_E [70, 110]. The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

3.4.5 Plate $2H$ – Ψ_B Set-3 Relaxed – Algorithm D_E

The Plate $2H$ tension test is formulated using the following scaling sequence for $n = 3, 4, 5, 6$:

$$\begin{aligned} H^{(n)}/h^{(n)} &= 11 \\ H^{(n)} &= \tfrac{1}{64} \\ \Omega^{(n)} &= [0, 2^n H] \times [0, 2^{(n-1)} H] \times [0, 2H] \, . \end{aligned}$$

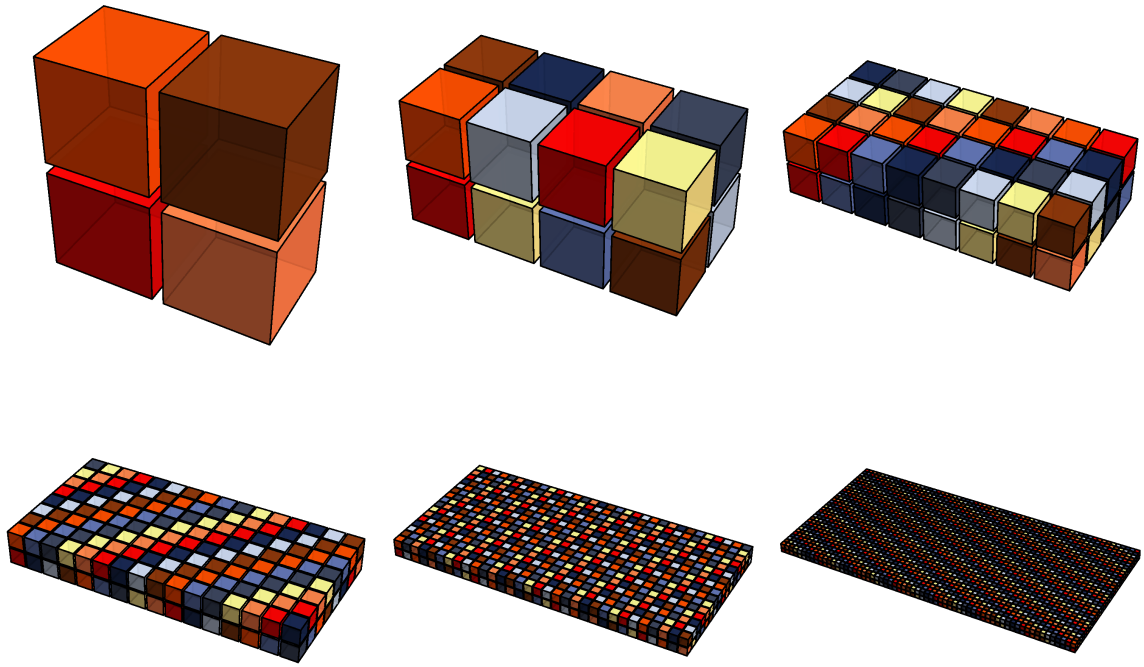


Figure 3.16: Weak scalability; Plate $2H$; $H/h = 11$; Sequence of FETI-DP subdomains; Scaling stages $1 \leq n \leq 6$.

Plate $2H$ – Solver Parameters and Tolerances

We have used equivalent parameters for all problems in the weak scalability sequence. For the largest problem in the scaling sequence using 4096 cores, we have used the following command string for `mparfeap`:

Command line options for `mparfeap`

```
aprun -n 4096
mparfeap-js-opt-mumps
-o=../edges_Hh11_psi_b_relaxed:4096:2012-10-10_19:16:19:1
-nw=4096
-nwpp=1
-stamp=edges_Hh11_psi_b_relaxed:4096:2012-10-10_19:16:19:1
-log_summary
-load_stepper:strategy=adaptive(n=512:initial=0.0:final=4.0:alpha=1.5:delay=2)
```

```
-load_stepper:use_linear_extrapolation=1
-newton:use_initial_guess_for_correction=1
-load_stepper:tecplot=0
-load_stepper:save_intermediate_solutions=0
-load_stepper:use_pseudo_time_for_dirichlet_boundary_interpolation=1
-newton:atol=1e-7
-newton:rtol=2e-14
-feti:symmetric=1
-ksp_type gmres
-ksp_gmres_modifiedgramschmidt
-ksp_rtol 2e-14
-ksp_atol 1e-9
-ksp_max_it 888
-ksp_gmres_restart 444
-ksp_monitor_singular_value
-use_rho_local
-use_local_Kcc
```

Plate $2H$ – Tabulated Results

Cores	Global Problem [# d.o.f]	Local Problem [# d.o.f]	Coarse Problem [# d.o.f]	Dual Problem [# d.o.f]
4096	134 101 575	36 501	48 102	14 277 717
1024	33 596 775	36 501	11 766	3 507 747
256	8 434 935	36 501	2 814	844 491
64	2 126 655	36 501	642	194 847

Table 3.8: Weak scalability; Plate $2H$; $H/h = 11$; Problem sizes.

Cores	Loading Attempts	Eff. [%]	Newton It. (Σ)	Eff. [%]	FETI-DP It. (Σ)	Eff. [%]	FETI-DP It. ($\oslash$)	Eff. [%]	FETI-DP Cond.* ($\oslash$)	Eff. [%]
64	24	100	48	100	688	100	14.33	100	22.15	**100**
256	24	100	61	79	966	71	15.84	91	24.43	**91**
1024	24	100	78	62	1 423	48	18.24	79	29.05	**76**
4096	29	83	95	51	1 935	36	20.59	70	33.50	**66**

Cores	FETI-DP ($\boldsymbol{\Sigma}$) [s]	Eff. [%]	FETI-DP ($\oslash$) [s]	Eff. [%]	GMRES ($\boldsymbol{\Sigma}$) [s]	Eff. [%]	GMRES ($\oslash$) [s]	Eff. [%]	Run [s]	Eff. [%]
64	2 195	100	45.73	100	312	100	6.50	100	2 854	**100**
256	2 983	74	48.90	94	522	60	8.56	76	3 843	**74**
1024	4 225	52	54.17	84	1 046	30	13.4	48	5 411	**53**
4096	7 496	29	78.91	58	3 158	10	33.2	20	10 410	**27**

Table 3.9: Weak Scalability; Plate $2H$; $H/h = 11$; Timings and efficiencies for parallel scalability; FETI-DP coarse space Algorithm D_E [70, 110]. The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

Plate $2H$ – Newton Iterations

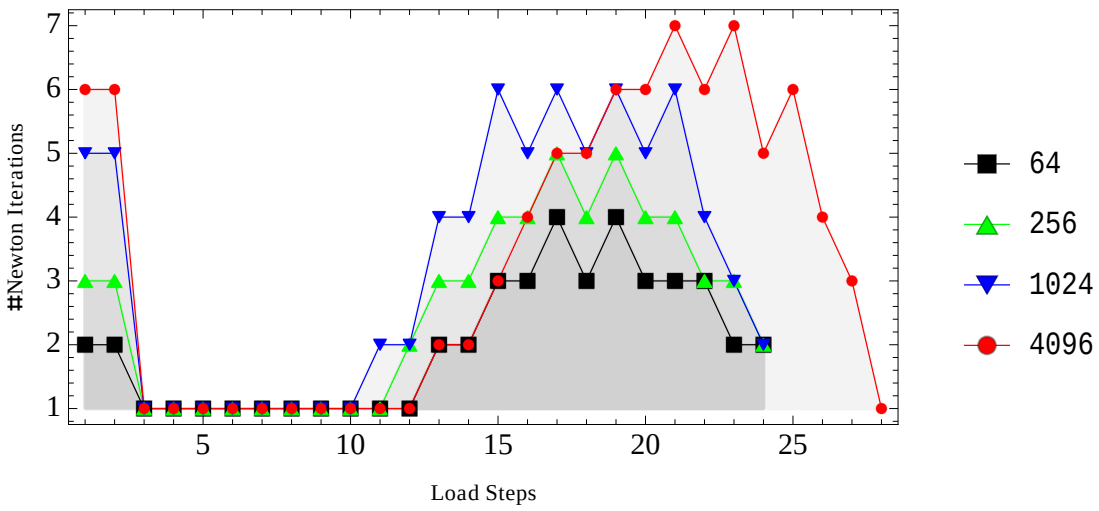


Figure 3.17: Weak scalability; Plate $2H$; $H/h = 11$; Newton iterations; The number of Newton iterations increases with the number of CPU cores. Thus the total number of FETI-DP systems to be solved increases and parallel scalability is adversely affected; see also §3.4.8.

Plate $2H$ – Numerical Scalability and Convergence Statistics

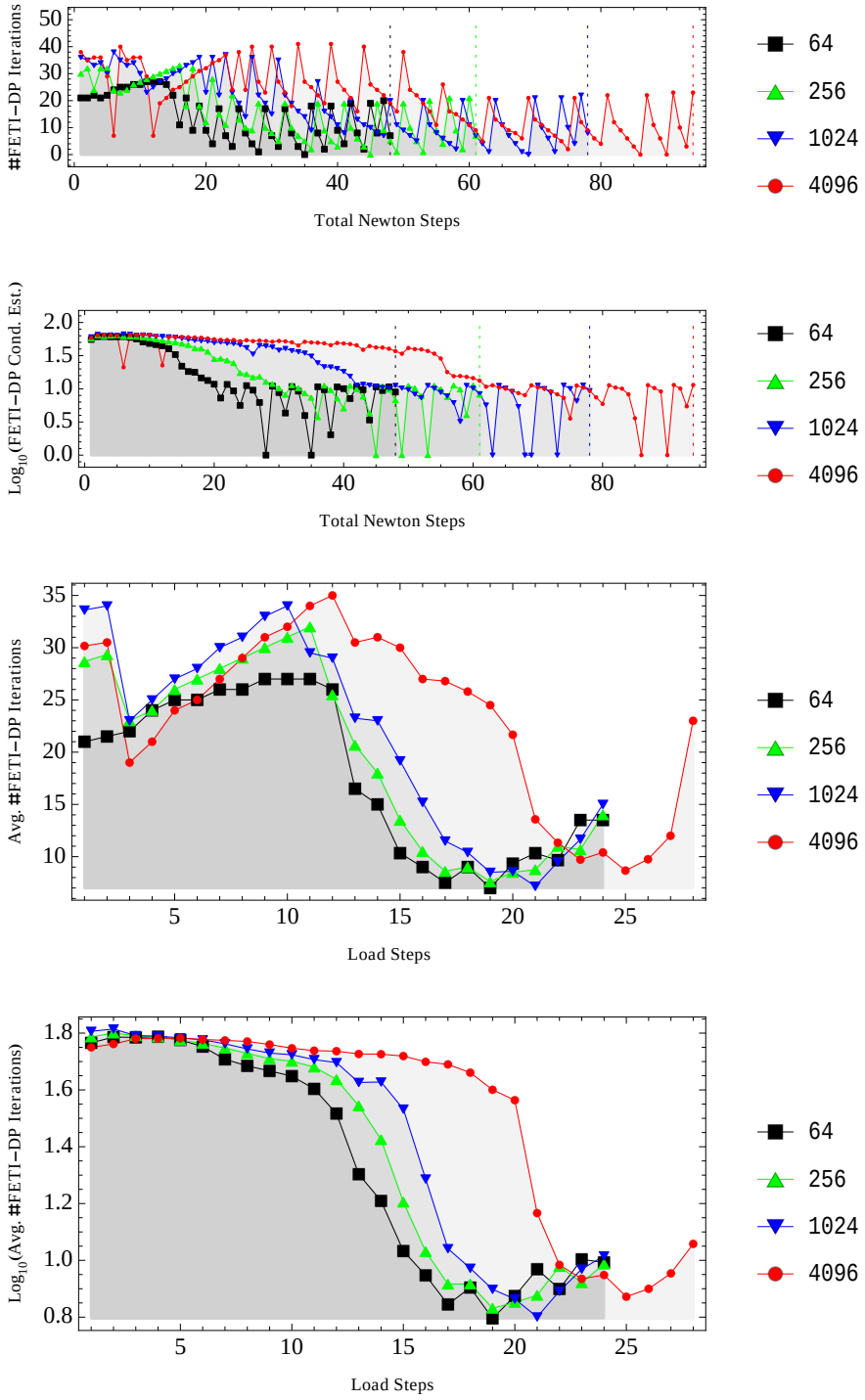


Figure 3.18: Weak scalability; Plate $2H$; $H/h = 11$; FETI-DP iterations and condition number estimates; FETI-DP coarse space Algorithm D_E [70, 110]. The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

Plate $2H$ – Parallel Scalability and Timing Statistics

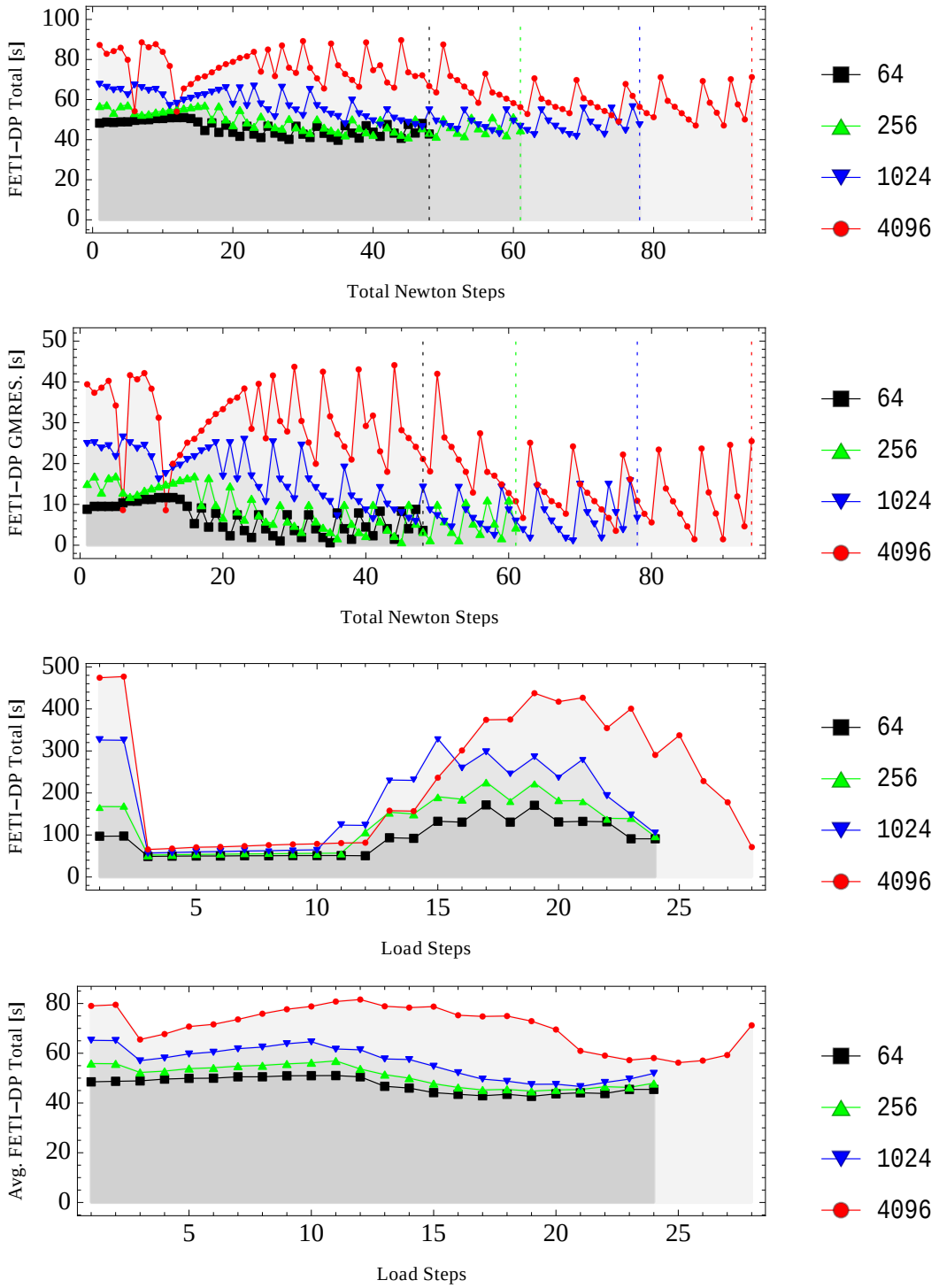


Figure 3.19: Weak scalability; Plate $2H$; $H/h = 11$; FETI-DP timings; FETI-DP coarse space Algorithm D_E [70, 110]. The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

3.4.6 Plate $4H$ – Ψ_B Set-3 Relaxed – Algorithm D_E

We consider the following computational domains and FETI-DP domain decompositions for scaling levels $n = 2, 3, 4, 5$:

$$\begin{aligned} H^{(n)}/h^{(n)} &= 13 \\ H^{(n)} &= \tfrac{1}{32} \\ \Omega^{(n)} &= [0, 2^n H] \times [0, 2^{(n-1)} H] \times [0, 4H] \ . \end{aligned}$$

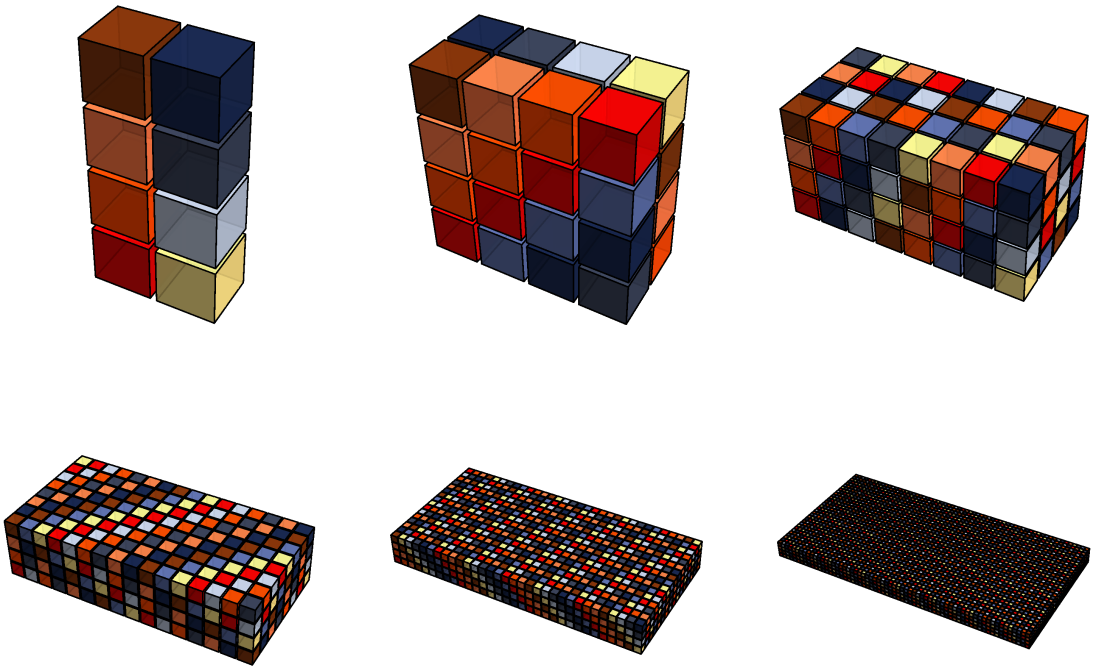


Figure 3.20: Weak scalability; Plate $4H$; $H/h = 13$; Sequence of partitioned meshes for a plate of height $4H$, $H = \frac{1}{32}$.

Plate $4H$ – Solver Parameters and Tolerances

We have used equivalent parameters for all problems in the weak scalability sequence. For the largest problem in the scaling sequence using 2048 cores, we have used the following command string for `mparfeap`:

Command line options for `mparfeap`

```
aprun -n 2048
mparfeap-js-opt-mumps
-o=../edges_Hh13_psi_b_relaxed_half_nodes:2048:2012-10-10_04:51:28:1
-nw=2048
-nwpp=1
-stamp=edges_Hh13_psi_b_relaxed_half_nodes:2048:2012-10-10_04:51:28:1
-log_summary
-load_stepper:strategy=adaptive(n=512:initial=0.0:final=4.0:alpha=1.5:delay=2)
```

```
-load_stepper:use_linear_extrapolation=1
-newton:use_initial_guess_for_correction=1
-load_stepper:tecplot=0
-load_stepper:save_intermediate_solutions=0
-load_stepper:use_pseudo_time_for_dirichlet_boundary_interpolation=1
-newton:atol=1e-7
-newton:rtol=2e-14
-feti:symmetric=1
-ksp_type gmres
-ksp_gmres_modifiedgramschmidt
-ksp_rtol 2e-14
-ksp_atol 1e-9
-ksp_gmres_restart 444
-ksp_max_it 888
-ksp_monitor_singular_value
-use_rho_local
-use_local_Kcc
```

Plate $4H$ – Tabulated Results

Each FETI-DP subdomain is mapped to one CPU core of the Cray XT6. The problem sizes for the considered scaling levels are:

Cores	**Global Problem [# d.o.f]**	**Local Problem [# d.o.f]**	**Coarse Problem [# d.o.f]**	**Dual Problem [# d.o.f]**
2048	109 418 715	59 049	21 168	10 725 237
512	27 453 195	59 049	5 208	2 591 517
128	6 912 675	59 049	1 260	602 865
32	1 752 975	59 049	294	128 091

Table 3.10: Weak scalability; Plate $4H$; $H/h = 13$; Problem sizes.

Cores	**Loading Attempts**	**Eff. [%]**	**Newton It.** (Σ)	**Eff. [%]**	**FETI-DP It.** (Σ)	**Eff. [%]**	**FETI-DP It.** ($\oslash$)	**Eff. [%]**	**FETI-DP Cond.*** ($\oslash$)	**Eff. [%]**
32	24	100	45	100	644	100	14.31	100	20.91	**100**
128	24	100	57	79	822	78	14.42	99	22.64	**92**
512	24	100	67	67	1108	58	16.54	87	26.05	**80**
2048	24	100	79	57	1642	39	20.78	69	35.10	**60**

Cores	**FETI-DP (Σ) [s]**	**Eff. [%]**	**FETI-DP ($\oslash$) [s]**	**Eff. [%]**	**GMRES (Σ) [s]**	**Eff. [%]**	**GMRES ($\oslash$) [s]**	**Eff.**	**Run [s]**	**Eff. [%]**
32	5 121	100	113.8	100	910	100	20.2	100	6 158	**100**
128	7 033	73	123.4	92	1288	71	22.6	89	8 341	**74**
512	9 083	56	135.6	84	1922	47	28.7	70	10 667	**58**
2048	12 950	40	163.9	69	3766	24	47.7	42	14 910	**41**

Table 3.11: Weak scalability; Plate $4H$; $H/h = 13$; Timings and efficencies for parallel scalability; FETI-DP coarse space Algorithm D_E [70, 110]. The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

Plate $4H$ – Newton Iterations

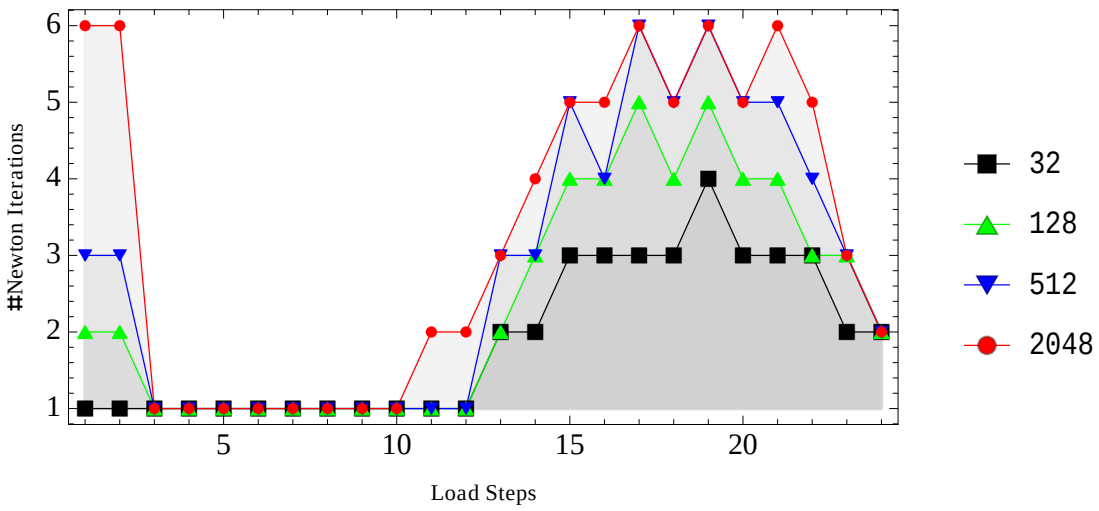


Figure 3.21: Weak scalability; Plate $4H$; $H/h = 13$; Newton iterations; The number of Newton iterations increases with the number of CPU cores. Thus the total number of FETI-DP systems to be solved increases and parallel scalability is adversely affected; see also §3.4.8.

Plate $4H$ – Numerical Scalability and Convergence Statistics

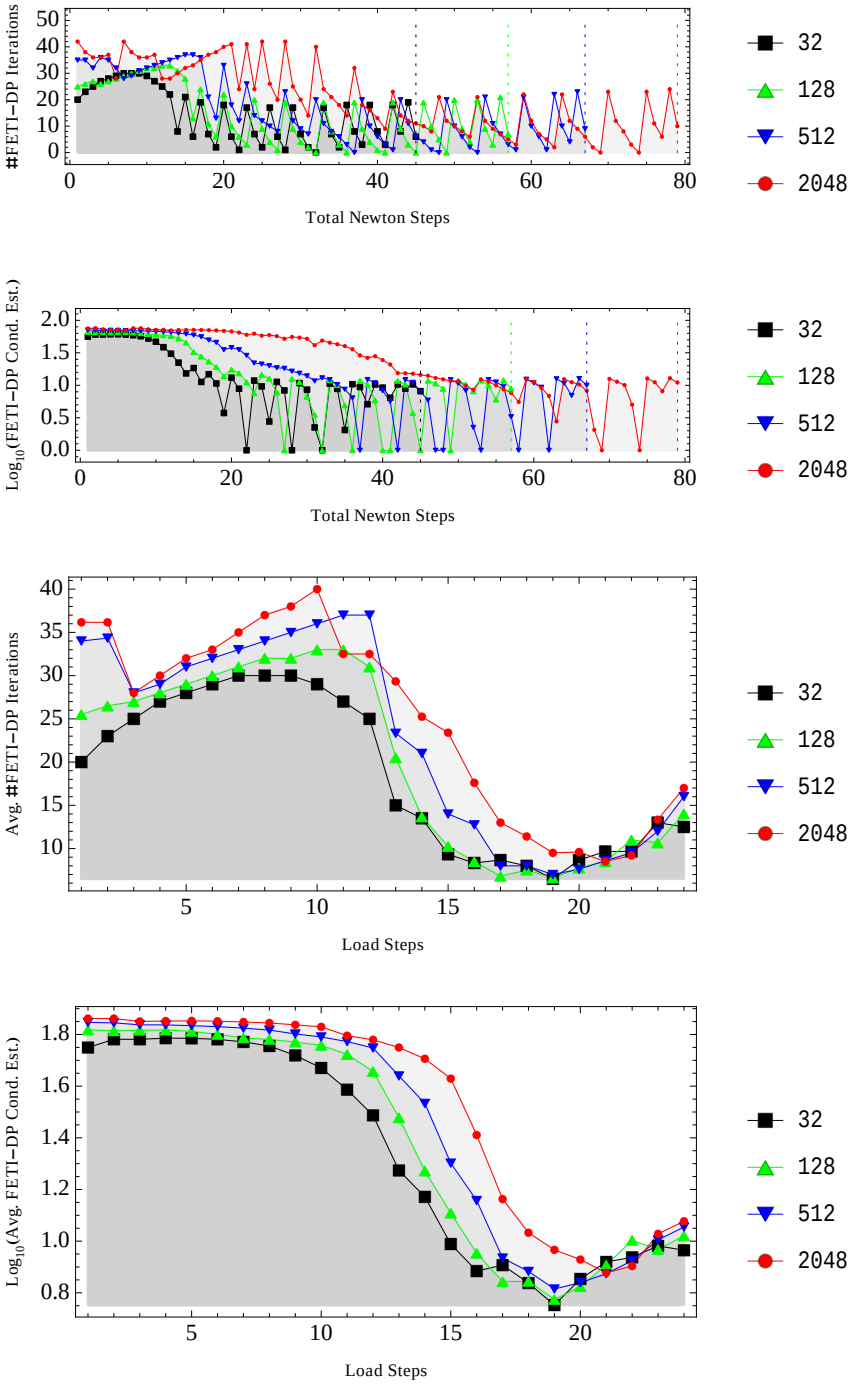


Figure 3.22: Weak scalability; Plate $4H$; $H/h = 13$; FETI-DP iterations and condition number estimates; FETI-DP coarse space Algorithm D_E [70, 110]. The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

Plate $4H$ – Parallel Scalability and Timing Statistics

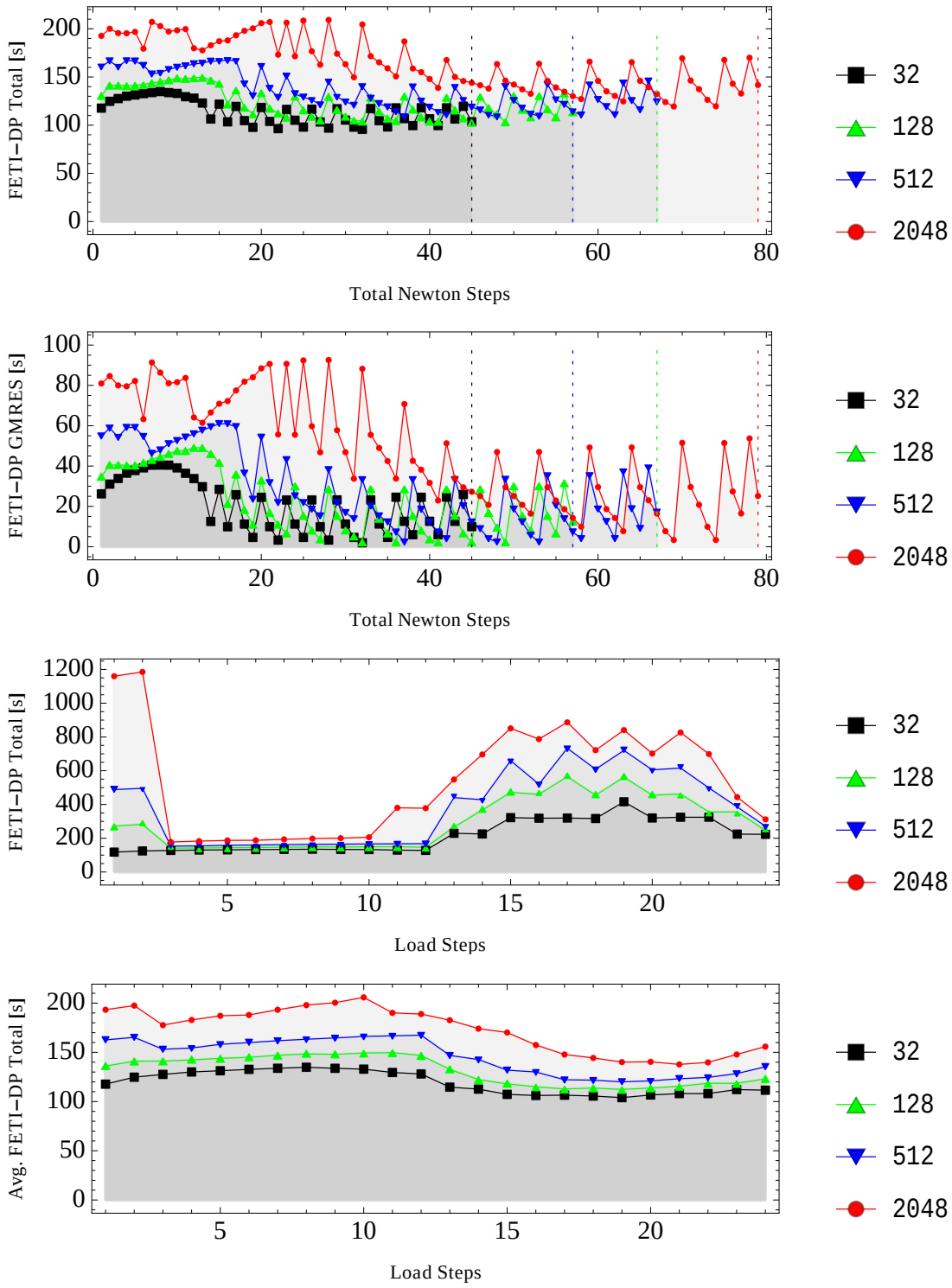


Figure 3.23: Weak scalability; Plate $4H$; $H/h = 13$; FETI-DP timings; FETI-DP coarse space Algorithm D_E [70, 110]. The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

3.4.7 Cube – Ψ_B Set-3 Relaxed – Algorithm D_E

We consider the computational domains and FETI-DP domain decompositions for the scaling levels $n = 1, 2, 3, 4$:

$$\begin{aligned} H^{(n)}/h^{(n)} &= 8 \\ H^{(n)} &= \tfrac{1}{2^n} \\ \Omega^{(n)} &= [0,1]^3 \end{aligned}$$

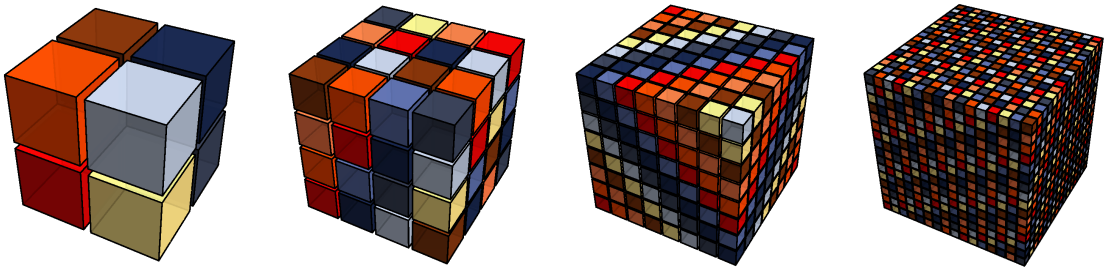


Figure 3.24: Weak scalability; Cube; $H/h = 8$; Sequence of partitioned meshes for a unit cube geometry

Cube – Solver Parameters and Tolerances

We have used equivalent parameters for all problems in the weak scalability sequence. For the largest problem in the scaling sequence using 4096 cores, we have used the following command string for `mparfeap`:

Command line options for `mparfeap`

```
aprun -n 4096
mparfeap-js-opt
-o=../edges_Hh8_psi_b_relaxed:le=1:ig=1:4096:2012-09-07_16:43:32:1
-nw=4096
-nwpp=1
-stamp=edges_Hh8_psi_b_relaxed:le=1:ig=1:4096:2012-09-07_16:43:32:1
-load_stepper:strategy=adaptive(n=512:initial=0.0:final=0.4:delay=2:alpha=1.5)
-load_stepper:use_linear_extrapolation=1
-newton:use_initial_guess_for_correction=1
-load_stepper:tecplot=0
-load_stepper:save_intermediate_solutions=0
-load_stepper:use_pseudo_time_for_dirichlet_boundary_interpolation=1
-ksp_type gmres
-ksp_gmres_modifiedgramschmidt
-ksp_rtol 2e-14
-ksp_atol 1e-9
-ksp_gmres_restart 444
-ksp_max_it 888
-ksp_monitor_singular_value
-use_rho_local
-use_local_Kcc
```

Note that in contrast to the previously presented weak scalability problems, the input data were prepared to compute a tension test with 10% deformation here. Correspondingly, the Dirichlet boundary data amount to 10% deformation at pseudo time $t = 1.0$. As this turned out to be

unrealistic due to time restrictions for the full Cray XT6m machine, we accomodated for this by setting the final pseudotime `final=0.4` which effectively results in a computation of a tension test with 4% deformation.

Cube – Tabulated Results

Cores	Global Problem [# d.o.f]	Local Problem [# d.o.f]	Coarse Problem [# d.o.f]	Dual Problem [# d.o.f]
4096	72 412 707	20 577	37 980	10 611 585
512	9 145 875	20 577	4 788	1 244 985
64	1 167 051	20 577	576	134 901
8	151 959	20 577	54	11 499

Table 3.12: Weak scalability; Cube; $H/h = 8$; Problem sizes.

Cores	Loading Attempts	Eff. [%]	Newton It. (Σ)	Eff. [%]	FETI-DP It. (Σ)	Eff. [%]	FETI-DP It. ($\oslash$)	Eff. [%]	FETI-DP Cond.* ($\oslash$)	Eff. [%]
8	24	100	74	100	1 141	100	15.42	100	13.28	**100**
64	24	100	85	87	1 525	75	17.94	86	18.15	**73**
512	24	100	103	72	1 839	62	17.85	86	20.36	**65**
4096	28	86	125	59	2 321	49	18.72	82	22.66	**59**

Cores	FETI-DP ($\boldsymbol{\Sigma}$) [s]	Eff. [%]	FETI-DP ($\oslash$) [s]	Eff. [%]	GMRES ($\boldsymbol{\Sigma}$) [s]	Eff. [%]	GMRES ($\oslash$) [s]	Eff. [%]	Run [s]	Eff. [%]
8	901	100	12.18	100	135	100	1.82	100	1273	**100**
64	1 255	72	14.76	83	208	65	2.44	75	1692	**75**
512	1 745	52	16.94	72	374	36	3.63	50	2309	**55**
4096	5 267	17	42.13	29	1264	11	10.1	18	6142	**21**

Table 3.13: Weak scalability; Cube; $H/h = 8$; Timings and efficiencies for parallel scalability; FETI-DP coarse space Algorithm D_E [70, 110]. The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

Cube – Newton Iterations

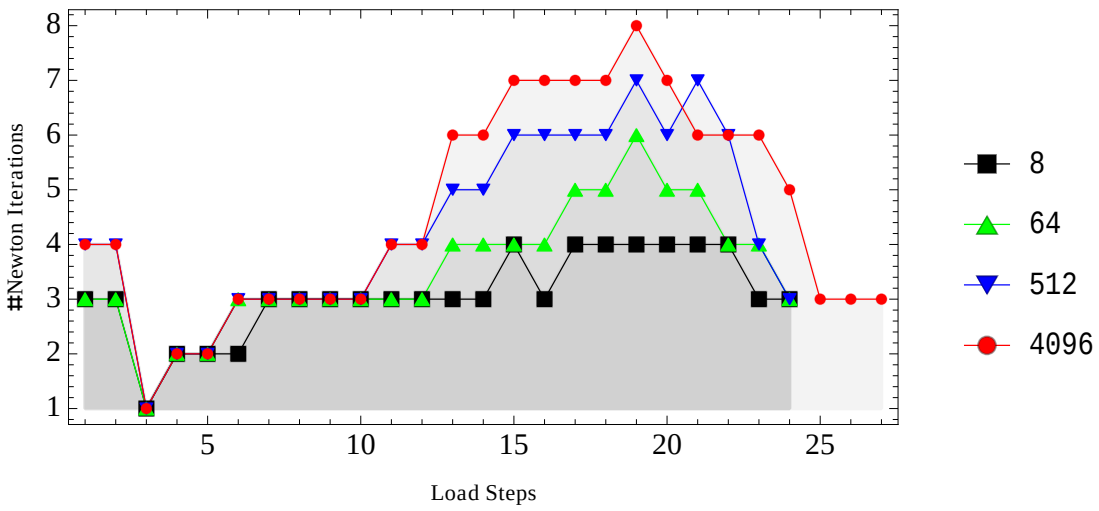


Figure 3.25: Weak scalability; Cube; $H/h = 8$; Newton iterations; The number of Newton iterations increases with the number of CPU cores. Thus the total number of FETI-DP systems to be solved increases and parallel scalability is adversely affected; see also §3.4.8.

Cube – Numerical Scalability and Convergence Statistics

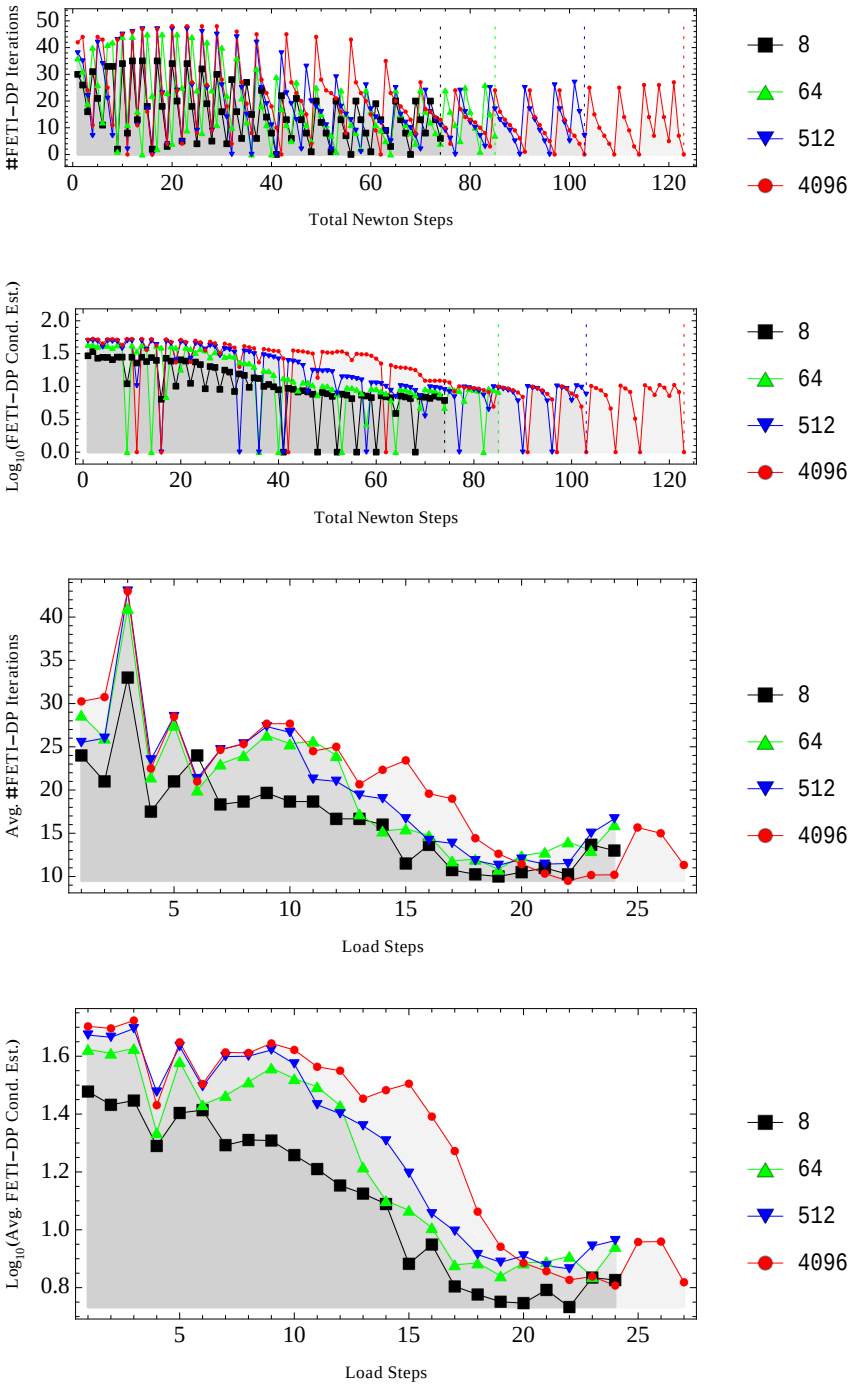


Figure 3.26: Weak scalability; Cube; $H/h = 8$; FETI-DP iterations and condition number estimates. FETI-DP coarse space Algorithm D_E [70, 110]. Note the increasing total number of Newton iterations. The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

Cube – Parallel Scalability and Timing Statistics

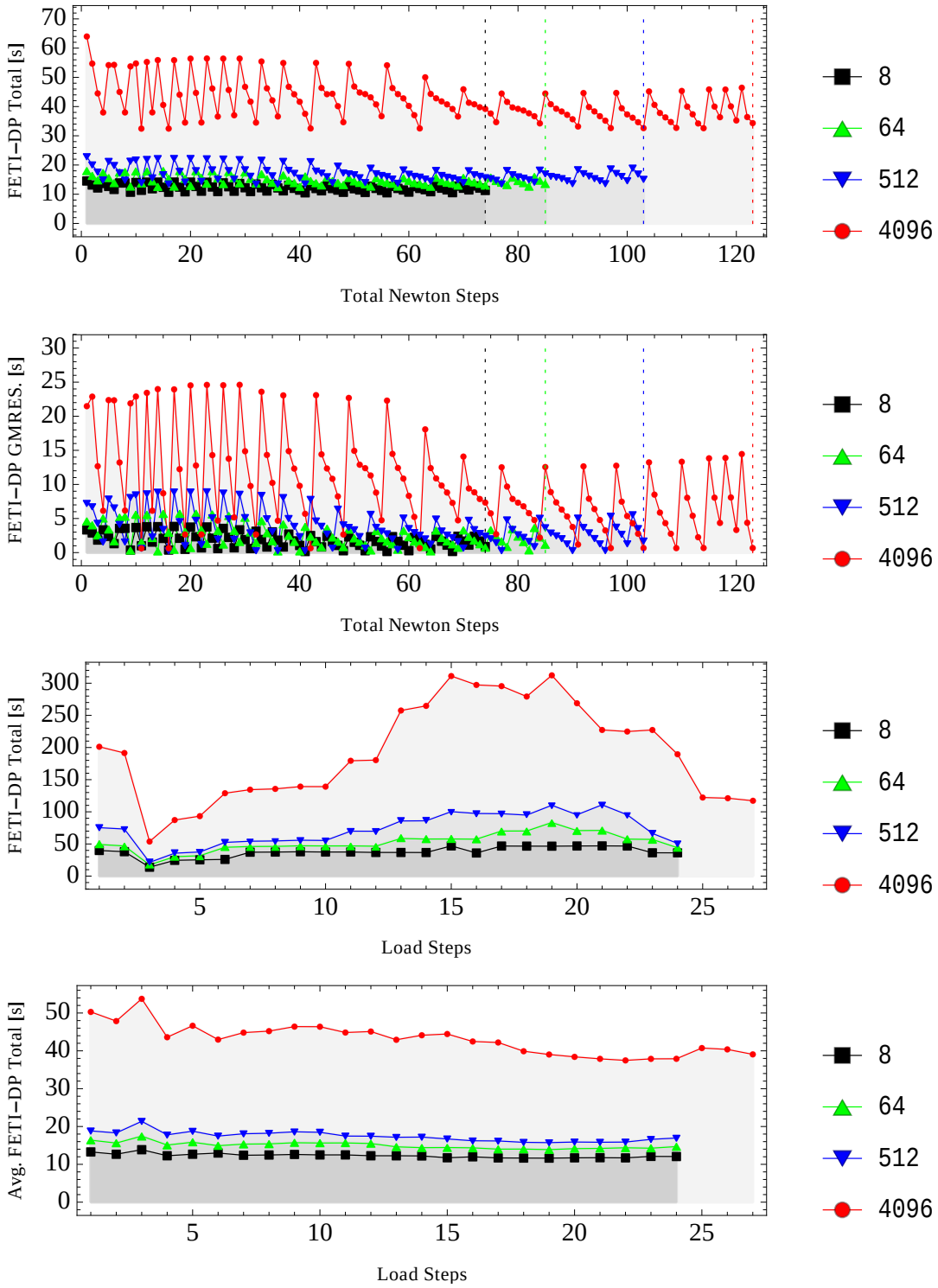


Figure 3.27: Weak scalability; Cube; $H/h = 8$; FETI-DP timings; FETI-DP coarse space Algorithm D_E [70, 110]. The lack of scalability is for a large part due to an increasing number of Newton iterations, see also §3.4.8.

3.4.8 Discussion

Let us now try to distill some observations and conclusions from the previously presented weak scalability experiments which we obtained with the Newton-Krylov-FETI-DP solver `mparfeap`.

Remark 3.4.2 (Numerical Scalability and Condition Number Estimates from GMRES)**.** *Numerical scalability of a preconditioning strategy for the weak formulation of a linear partial differential equation is in general shown by proving a priori eigenvalue bounds for suitable finite element discretized problems. One is then interested in the implied condition number bounds for the full and preconditioned discretized linear systems. For numerically scalable algorithms, the a priori condition number bound for the preconditioned system may only depend weakly, i.e., logarithmically, on the inverse of the typical finite element diameter $1/h$. In domain decomposition methods the weak dependency is usually on H/h. For example, for FETI-DP with the Dirichlet preconditioner, we have the typical estimate, c.f. §1.3:*

$$\kappa(\mathfrak{M}_D^{-1} \cdot \mathfrak{F}) = \kappa(B_{D,\Gamma}\widetilde{\mathcal{S}}B_{D,\Gamma}^T \cdot B_\Gamma \widetilde{\mathcal{S}}^{-1} B_\Gamma^T) \leq C\,(1 + \log(H/h))^2. \tag{3.4.1}$$

Such bounds have been obtained only for certain model problems, e.g, linear elasticity with an appropriate coarse space in three dimensions, see e.g., Klawonn and Widlund [76]. For a conjugate gradient method (CG) this bound has immediate consequences on the number of iterations until convergence. For GMRES, however, this relation is not completely understood, see e.g., Klawonn [68], and Klawonn and Rheinbach [71] for related discussions in the context of FETI-methods. Moreover, in our applications in biomechanics, no a priori convergence bounds are known for the linear systems arising from linearization in the Newton-Krylov-FETI-DP algorithm. In our presented computations, we have recorded the condition number estimates computed by the Arnoldi process in the `PETSc` *GMRES solver. This is indicated by the column heading Cond.** *and is not necessarily an accurate estimate. We summarize that the provided condition number estimates are a subtle indicator for numerical scalability and have to be interpreted with some care.*

Parallel Scalability of the Assembly Phase

A central motivation for the development of the presented solver framework was the sequential assembly phase in the previously developed semi-parallel framework. This phase was fundamentally non-scalable. This imposed serious limitations for the global problem sizes. From the timing data recorded during our weak scalability experiments, we deduce that the assembly phase in `mparfeap` is scaling near perfectly. Note that this is expected, as there is no coupling between the assembly in different subdomains, i.e., finite element assembly is trivially parallel.

Thus, the larger the problem size, the more time is spent in the linear FETI-DP solver while the share of the runtime spent in the assembly phase decreases accordingly. In the presented experiments the linear solver consumed up to 86.5% of the total simulation time (plate $4H$). This clearly illustrates the need for highly efficient linear solvers, such as FETI-DP, for Newton-Krylov approaches. Linear solver performance is obviously the dominating factor and key to push the limits for the efficient solution of large-scale nonlinear problems based on Newton-Krylov methods.

Weak Nonlinear Scalability and Choice of Convergence Criteria

The lack of scalability is mostly due to an increase of Newton iterations. This is easily observed by comparing the number of Newton iterations for the different experiments in Table 3.28. We are by no means the first to observe this phenomenon. E.g., in their award winning paper Adams, Bayraktar, Keaveny, and Papadopoulos [1] observed a consistent increase of Newton iterations from 5 to 6 from the coarsest to the finest discretization of their model and remarked “that the nonlinear problems are becoming more challenging as the mesh is refined”.

This seems to be expected to a certain degree due to the h-refinement induced by the weak scaling sequence. If we consider the number of Newton iterations as a function of h, then it is *asymptotically bounded* in the framework described in Deuflhard [34, Chapter 8]. The asymptotic bound for the Newton iterations depends on the nonlinear functional, the starting iterate, the function space norms and the convergence criteria. Thus, assuming that the balance equation fits the framework for the analysis in Deuflhard [34, Chapter 8], the number of Newton iterations is to be expected to change on h-refinement until the asymptotic bound is finally reached. The number of Newton iterations will then simply change accordingly for the different stages of the scaling ladder used in a weak scaling sequence.

Since this is influenced by the choice of convergence criteria and the choice of norms considered in the outer Newton and inner Krylov iterations of a Newton-Krylov-scheme we state our convergence criteria in some more detail. To formulate the convergence criteria, we have resorted to the Euclidean norms of the residuals of the coefficient vectors

$$||r_{\text{Newton}}(u)||_2 \ : \mathbb{R}^{\#\text{d.o.f.}} \rightarrow \mathbb{R}_0^+ \qquad \text{and} \tag{3.4.2}$$

$$||r_{\text{FETI-DP}}(\lambda)||_2 \ : \mathbb{R}^{\#\text{dual d.o.f.}} \rightarrow \mathbb{R}_0^+ \ . \tag{3.4.3}$$

We refer to these as the nonlinear and linear residuals norms. This choice is not ideal, as this family of norms depends on the number of degrees of freedom. On the other hand, these norms are a standard choice used by default, e.g., in PETSc. We document here, what is the effect on our weak scalability experiments using absolute convergence criteria.

From a numerical analysis standpoint, it is clear that the use of appropriate function space norms, such as Sobolev norms, would have been more appropriate for studying convergence for $h \rightarrow 0$. The norm itself is then independent of h. However, in our particular software context, see Chapter 2, an implementation of an appropriate function space norm was unfortunately more work than we could invest. This might have helped, but even for well matched tolerances and function space norms for the inner and outer iterations, it seems that an increasing total number of Newton iterations would still to be expected. This is our interpretation of Table 8.2 in Deuflhard [34, Chapter 8,p.388]. Note that the convergence criteria used there are different from ours. It would be interesting to see how this table would continue for very high resolutions, as presented here. Although it seems that the number of Newton iterations has already reached the asymptotic limit for some of the experiments, this is not necessarily the case. The number of Newton iterations might also continue to grow very slowly.

Let us conclude: *The scalability properties of the linear Krylov-solver are not automatically inherited by the corresponding nonlinear Newton-Krylov method.* Even a perfectly scaling parallel linear iterative solver will not yield a parallel scalable Newton-Krylov-method *unless the number of load steps and Newton steps stays constant for all levels of the scaling sequence.* This is certainly the case, if and only if an asymptotic limit is reached. We consider it thus rather unrealistic to achieve efficiencies for the full nonlinear scalability which are comparable to those obtained for simple linear model problems. These might however be achieved in the average per Newton-step, if the preconditioner for the linearized systems is working effectively.

Efficiency Measures for Nonlinear Scalability

The increasing number of Newton iterations affects the measures for numerical and parallel scalability.

At multiple places, we have plotted and tabulated quantities in the average per load step or Newton iteration. Most prominently, we have tabulated *efficiencies in the average per Newton iteration* for numerical and parallel scalability. We consider these as convenient measures for the numerical

and parallel scalability for Newton-Krylov solvers. These measures are more stable with respect to the increasing number of Newton iterations. These data yield information on the scalability of the FETI-DP method for all encountered linear systems (averages) and on the nonlinear scalability for the nonlinear Newton-Krylov-FETI-DP solver (totals).

Let us shortly mention a possible shortcoming of the setup of our scaling sequences. For our presented experiments, we have strived for equivalent loading curves across the full scaling ladder, i.e., we applied the same load stepping strategy on each scaling level. This is not ideal for the smaller problems for which a smaller number of load steps might have been sufficient. Currently, we avoided the effort to determine the optimal load stepping curve for each scaling level separately.

Note that weak nonlinear scalability could be positively affected by a choice of initial values for the Newton scheme obtained from mesh sequencing, i.e., by interpolation of solutions obtained using a coarser mesh. Typically in structural mechanics applications, there is no access to coarser mesh resolutions and in this situation this strategy is not feasible.

In all presented tables, we note an increase of the number of FETI-DP iterations in the average per Newton-step. This may be due to the anisotropy or the incompressibility constraint. Note that in the current algorithm, a coarse space designed for compressible linear elasticity has been used, i.e., FETI-DP algorithm D_E. In the context of linear elasticity, adapted coarse spaces have been developed, e.g., by Klawonn, Rheinbach, and Wohlmuth [82] and Li and Widlund [89]. See also Gippert, Klawonn, and Rheinbach [57], the dissertation by Gippert [55] and references therein. In particular, we want to mention the contribution by Dohrmann and LeHoucq [37].

Certainly, the integration of adapted coarse spaces into the present software framework is the next step.

Relative and Absolute Convergence Criteria and Robustness

A qualitative observation which we have consistently made during our experiments is that the usual relative termination criteria, e.g., `rtol` in `PETSc`, which use the relative residual norms are too unreliable for the hard problems here investigated. In general, these criteria lead to corrupted solutions whenever there is a spike in the sequence of initial residuals. This observation holds for the linear $r_{\text{FETI-DP}}(\lambda)$ and for the nonlinear residual $r_{\text{Newton}}(u)$ as well.

In the convergence history for the weak scalability experiments here presented, each and every linear FETI-DP solve converged due to an absolute convergence criterion for the coefficient vector u in the $||\cdot||_2$. Typically, the weak scaling efficiency ratios are less favorable for the choice of an absolute tolerance for the convergence criterion. However, in the unstructured case for realistic problems, relative tolerances have appeared to be of little practical use, since the initial residuals easily explode if the last load increment was too large. [1] Strong spikes are common to appear in the sequence of initial FETI-DP residual norms and usually trigger convergence leading to corrupt Newton corrections Δu. More precisely, the absolute tolerance of the FETI-DP solver is then so large, that the Newton correction is not even continuous across the subdomain interface anymore. Clearly, this is to be avoided and absolute convergence criteria do present a remedy. Let us stress once more that the parallel scalability is usually negatively impacted by a fixed absolute tolerance for the convergence of the Krylov-method which were necessary to achieve a reasonable robustness.

[1] The exceptional floating point numbers `INF` (infinity) and `NaN` (not a number) are encountered in practice and have to be handled appropriately.

Direct Solver Performance and Memory Requirements

A FETI-DP solver internally uses direct solvers at multiple places to factor process-local matrices in parallel.[2] These factorization phases are commonly referred to as "local solves". Somewhat counter-intuitively, choosing *a slower direct solver for the local solves usually results in better efficiency ratios*. But clearly, a larger share of the execution runtime is spent in a perfectly parallel program phase (no communication) and this improves scalability. The trade-off is a longer total application runtime.

During scalability testing on the Cray XT6m, it was important to minimize the total computation time by selecting the fastest linear direct solver for the local solves in FETI-DP. Note that the full machine was only available during special scaling time slots. All running computations were automatically killed by the job scheduler at the end of that given time slot which is a standard practice.[3]

We have found that MUMPS, compiled with compiler optimizations turned on and using the `INTEL Fortran`-compiler, is a very efficient and robust choice. As is well-known, direct solvers are very memory intensive. Exploitation of symmetric matrix storage in the `PETSc` FETI-DP implementation was a crucial step to achieve good results. The choice of Algorithm D_E for the weak scalability experiments was also motivated by the small memory footprint for the associated coarse problem. The use of larger coarse spaces, e.g., of Algorithm C, would be of interest and even preferred. This is, however, subject to future work since the factorization of the corresponding coarse problem is currently *too* memory intensive. For exact FETI-DP methods this could be improved, e.g., by sharing factorizations common to all processes in the FETI-DP solver on the compute node level. Another possible approach is to switch to inexact FETI methods, such as irFETI-DP. In that case the factorizations using direct solvers are replaced by iterative schemes which reduces the memory footprint enormously but at the same time the robustness.

[2] This is different for inexact FETI-DP methods and the discussion does not apply to these methods.
[3] Actually, there was a kind soul who magically kept some important jobs alive.

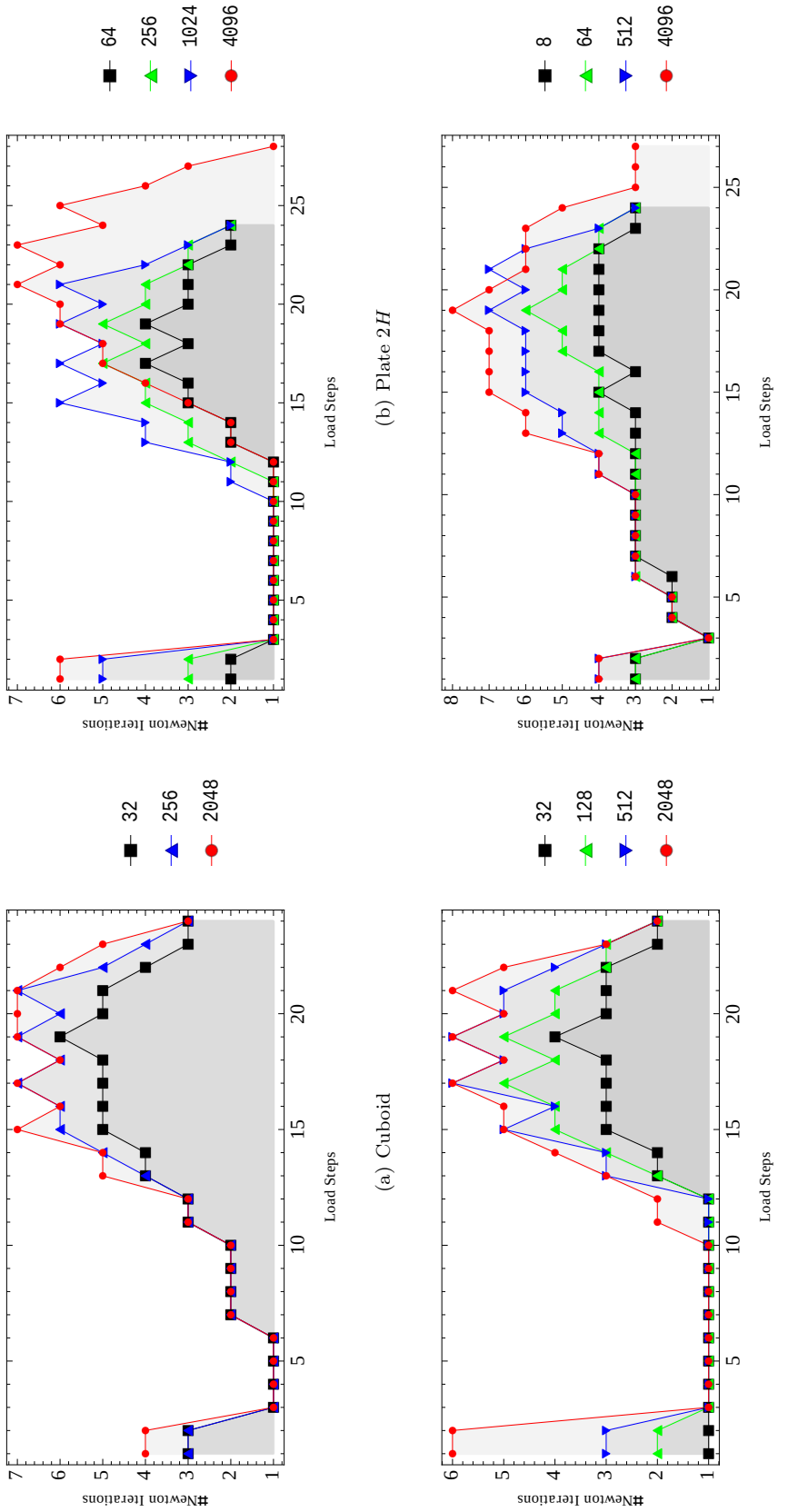
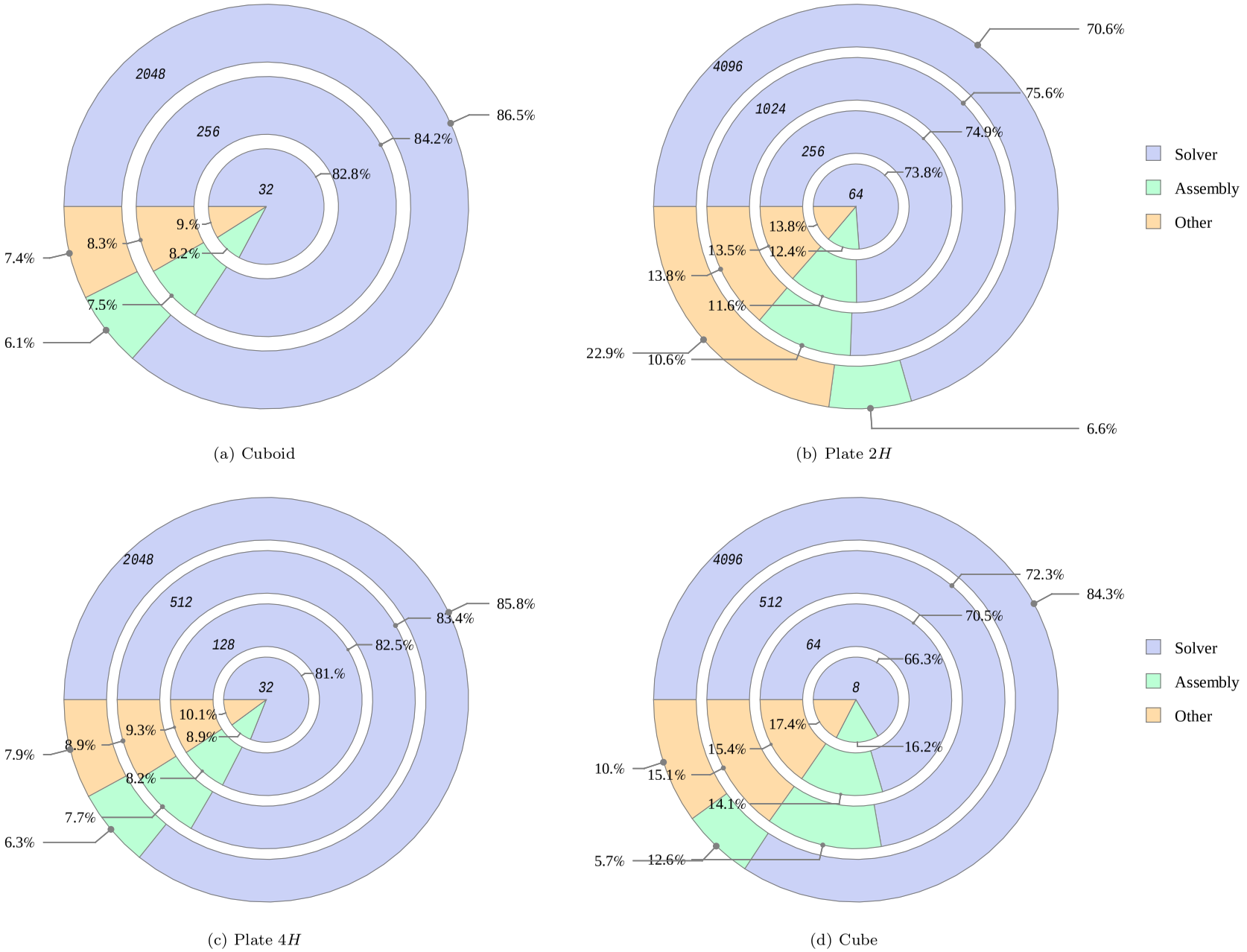


Figure 3.28: Weak Scalability; Comparison of Newton iteration counts. The number of Newton iterations consistently increases with the number of CPU cores. This has a negative impact on the nonlinear scalability of the Newton-Krylov-FETI-DP solver.



(a) Cuboid

(b) Plate $2H$

(c) Plate $4H$

(d) Cube

Figure 3.29: Application runtime shares in percent; The assembly phases are scaling almost perfectly; Up to 86.5% of the total runtime are solver time; For the largest data set of the Plate $2H$ sequence the disk I/O (Other) failed to scale well to 4096 processes.

3.5 Effectivity of Tuning Newton-Krylov-FETI-DP in Load Stepping Strategies

For the presented applications, it was *unavoidable* to tune the simulations with respect to the memory requirements and with respect to the total application runtime on the Cray XT6m. Only the latter aspect shall concern us here. In §1.4 we have introduced two strategies implemented by the software framework presented in Chapter 2: linear extrapolation for the displacement u and a non-zero initial guess for λ to start FETI-DP/GMRES. In this section we want to illustrate the gains we have obtained in scenarios comparable to the previously presented parallel simulations.

The linear extrapolation method for the displacement was proposed to us by J. Schroeder [113] and it is easily seen to be remarkably effective in our weak scalability tests. In fact, *all* of the weak scalability experiments presented in §3.4 profit a lot from the linear extrapolation feature. In the tension tests for cuboidal domains with Dirichlet boundary data, we have achieved a nice reduction of the number of load steps in conjunction with the adaptive load stepping strategy. For the exposition of the effects, we have compiled tables which are similar in style to those presented in Adams, Bayraktar, Keaveny, and Papadopoulos [1].

Let us mention first some related work: P. Gosselet and C. Rey investigated a selective reuse of Krylov subspaces in [58]. Their approach is related to our choice of a dual initial guess for FETI-DP. The efficiency of their proposed reusal strategy in our context would have been quite interesting to include in the presented comparisons that follow. One aspect is, however, easy to anticipate. The usage of the last dual solution as an initial guess for λ is trivial to implement and can be quite effective without incurring any cost.

3.5.1 First Level of the Cuboid Tension Test

Let us revisit the experiment for the Cuboid geometry described in §3.4.3. The effect of the linear extrapolation is extreme for this example to the extent that it was unrealistic to run that comparison for more than the smallest problem size of the weak scalability sequence before the feature was implemented. It is immediate to infer from Table 3.14 that the computations are sped up nearly by a factor of 12 when linear extrapolation for the displacement u is enabled. Note that this is an extreme example and that the convergence *can* also be negatively affected in some cases. For example, model artery 2 failed to converge with the linear extrapolation feature enabled.

#Cores	#D.O.F	Linear Extrapolation for u	Initial Guess for λ	Newton-It (total/failed)	FETI-DP-It (⌀)	Time
32	1 752 975	**yes**	**yes**	**78/0**	**19.90**	**185 m**
32	1 752 975	yes	no	78/0	32.44	205 m
32	1 752 975	no	yes	677/28	22.34	2 190 m
32	1 752 975	no	no	677/28	30.89	2 302 m

Table 3.14: Comparison matrix for the effect of linear extrapolation for u and the nonzero initial guess $\lambda^{(n-1)}$ for λ; The tension test for the Cuboid experiment is quite sensitive; Linear extrapolation speeds up the computation nearly by a factor 12; Adaptive load stepping strategy $\alpha = 1.5, \beta = 0.5$ and delay $d_{\text{step}} = 2$; Divergent Newton steps failed due to the exceeded limit of 888 GMRES iterations

	32 XT6 Cores (1.75 million d.o.f.)						2048 XT6 Cores (109 million d.o.f.)						
Newton step	**1**	**2**	**3**	**4**	**5**	**6**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Load step 1	40	34	19				52	51	47	26			
Load step 2	43	31	17				53	51	47	27			
Load step 3	45						48						
Load step 4	46						50						
Load step 5	47						51						
Load step 6	48						52						
Load step 7	48	23					53	25					
Load step 8	48	24					53	26					
Load step 9	47	27					54	30					
Load step 10	45	25					54	29					
Load step 11	41	24	11				54	32	19				
Load step 12	38	21	10				54	32	22				
Load step 13	34	20	13	4			53	33	29	24	17		
Load step 14	31	18	12	4			53	32	27	24	19		
Load step 15	28	17	12	8	1		50	32	27	24	20	15	6
Load step 16	27	16	11	8	0		43	26	24	20	16	10	
Load step 17	26	16	12	9	5		32	20	18	17	15	13	10
Load step 18	26	16	11	7	2		27	17	15	14	12	9	
Load step 19	25	16	12	9	5	0	26	17	14	13	11	8	5
Load step 20	25	16	12	8	4		26	17	13	11	8	5	0
Load step 21	26	17	12	8	3		27	18	14	12	9	7	3
Load step 22	26	16	11	5			27	17	13	9	6	1	
Load step 23	26	16	10				28	17	12	7	1		
Load step 24	27	15	6				29	16	7				
Time	**185 m**						**311 m**						

Table 3.15: Material model Ψ_B Set-3 relaxed; Load steps vs. Newton steps for the Cuboid tension test; The number of FETI-DP-iterations decreases in every Newton step due to the favorable initial guess $\lambda^{(n-1)}$; Coarsest and finest levels of the scaling sequence; A 0 indicates immediate convergence as a result of a perfect dual initial guess. Empty cells indicate prior convergence of the Newton scheme; Convergence criteria: $\text{atol}_{\text{Newton}} = 10^{-7}$ and $\text{atol}_{\text{FETI-DP}} = 10^{-9}$.

32 XT6 Cores (1.75 million d.o.f.)					
1	**2**	**3**	**4**	**5**	**6**
40	41	40			
43	43	43			
45					
46					
47					
48					
48	48				
48	48				
47	47				
45	45				
41	42	42			
38	38	38			
34	34	34	34		
31	32	32	32		
28	29	29	29	29	
27	27	27	27	27	
26	27	27	27	27	
26	26	26	26	26	
25	26	26	26	26	26
25	25	25	25	25	
26	26	26	26	26	
26	26	26	26		
26	26	26			
27	27	27			
205 m					

Table 3.16: For the setup compare with Tab. 3.15; Number of FETI-DP iterations using a *zero* initial guess for λ to start FETI-DP/GMRES. The number of GMRES iterations stays nearly constant.

3.5.2 Model Artery 2 – Ψ_B Set-3 Relaxed – Initial Loading

Let us turn our focus now over to model artery 2 with 13 million degrees of freedom to investigate the effects of the tuning options in a more complex scenario. For this experiment, we have used the stored energy function Ψ_B Set-3 relaxed, a formulation with relaxed volume penalty. For model artery 2, the linear extrapolation feature only converged for the initial loading phase.

Thus, the comparison matrix for the two tuning strategies discussed here only shows the initial loading phase of the arterial wall structure. In other words, we have not used physiological boundary conditions. Instead the computation was run with a final 1 [kPa] for the interior normal pressure and both sidecaps of the arterial wall structure model were clamped in all space directions using Dirichlet boundary conditions, since the more appropriate sliding boundary conditions were implemented only later.

Tuning Strategy	**None**				**Initial Guess for λ**				**Linear Extrapolation**				**Linear Extrapolation + Initial Guess for λ**			
Newton step	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
Load step 1	263	263			263	197			263	263			263	197		
Load step 2	270	270			270	196			270	270			270	196		
Load step 3	276	276			276	205			276	276			276	152		
Load step 4	280	279	279		280	206	142		279	279			279	154		
Load step 5	283	282	283		283	217	165		282	283			282	175		
Load step 6	285	285	285		285	220	165		285	285			285	180		
Load step 7	288	288	288		288	230	182		288	288			288	193		
Load step 8	290	289	290		290	230	181		291	290			291	197		
Load step 9	293	293	292	292	293	239	197	108	297	292	292		297	208	135	
Load step 10	294	294	294		294	237	194		306	294	294		306	209	134	
Load step 11	296	311	296	296	296	260	208	128	418	296	296		418	220	151	
Load step 12	298	311	298	298	298	256	206	128	507	298	298		507	222	155	
Load step 13	300	327	299	299	300	272	210	147	650	299	299	299	650	225	168	120
Time	**10h 21m**				**8h 45m**				**8h 50m**				**7h 24m**			

Table 3.17: Independent and combined analysis of linear extrapolation for u and $\lambda^{(n-1)}$ for λ to start FETI-DP/GMRES; Model artery 2; 13 million degrees of freedom; Synthetic boundary conditions: the artery is clamped on both caps; Interior normal pressure only up to 1 [kPa]; Both strategies interact favorably and can be successfully combined.

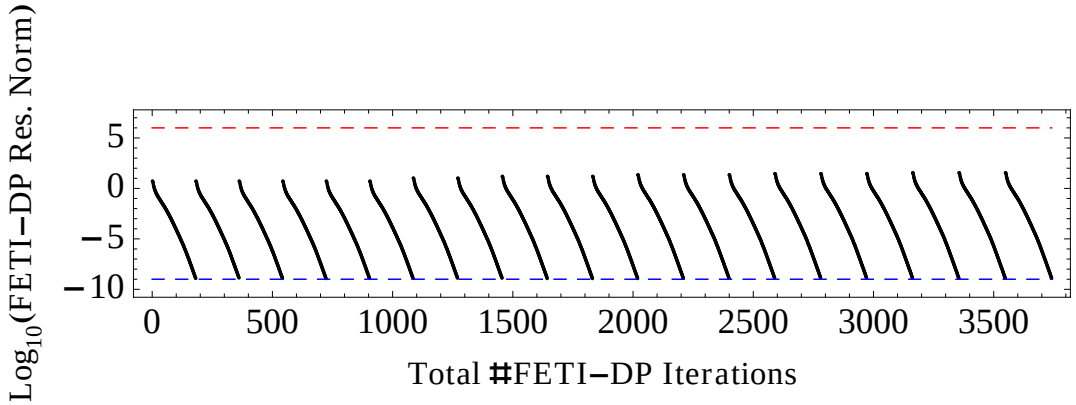
3.5.3 Model Artery 2 – Ψ_A Set-2 – Physiological Loading

We have run the setup for the strong scalability benchmark for model artery 2 described in §3.3 and compare two runs with the dual initial guess for λ turned on and off, respectively. We have used 512 processors on the Cray XT6m for this computation. The effect on the FETI-DP residual norms is displayed in Figure 3.30. Further, we have computed the quantitative computational savings in the following table:

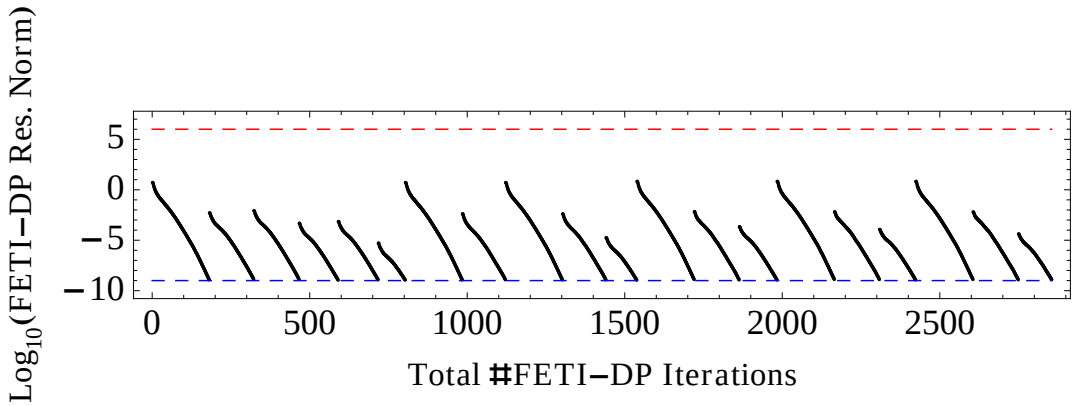
Initial Guess Strategy	**Total #FETI-DP Iterations**	**Solver [s]**	**Runtime [s]**
Zero	80 484	110 962	117 152
$\lambda^{(n-1)}$	57 204	84 152	89 287
Reduction	28.93%	24.16%	**23.78%**

Table 3.18: Computation of arterial wall stresses at 500 [mmHg]; Iteration and runtime reductions due to the choice of $\lambda^{(n-1)}$ as a non-zero initial guess for FETI-DP in each Newton step.

The last dual solution $\lambda^{(n-1)}$ is clearly an effective initial guess. It progressively reduces the number of FETI-DP iterations during Newton iteration. In our experience this particular strategy has shown good robustness and although counterexamples *may* exist, we have not encountered any so far. A 23.78% reduction of the total runtime is definitely a nice gain *at no cost*.



(a) Zero initial guess for $\lambda^{(n)}$



(b) Initial guess $\lambda^{(n-1)}$ for $\lambda^{(n)}$

Figure 3.30: Comparison of FETI-DP residual norms for the two different choices of initial guesses for FETI-DP; Model artery 2; Simulation with physiological boundary conditions; Gains for the displayed first six load steps; 3741 FETI-DP iterations in (a) vs. 2855 starting from $\lambda^{(n-1)}$ (b); Roughly 25% GMRES-iterations are saved **at no costs**.

3.6 Outlook

Finally, we want to present some currently ongoing as well as possible future developments and extensions.

The previously available simulations of arterial wall stress states in the physiological regime, obtained, e.g., in Brands, Klawonn, Rheinbach, and Schröder [23] and subsequent works, were based on the same semi-parallel framework. They were inherently non-scalable. This hurdle has finally been overcome using the parallel scalable software framework presented in Chapter 2. This new stepping stone was essentially reached through the development of `libfw`, or more precisely `libfw-js`, as a key component allowing for massively parallel assembly of the material models implemented by J. Schröders group in `FEAP-JS`.

Among, obviously, many possible directions of extensions, we have selected the following for a short overview over ongoing and possible future developments related to the present work:

Fluid-Structure Interaction (FSI): A very interesting current development is the usage of the material models available in `FEAP-JS` encapuslated by the contributed `libfw-js` for FSI simulations in the physiological regime and of a new degree of realism regarding the modelling of the arterial walls. This is the main theme of a joint DFG/SNF D-A-CH project:

Domain-Decomposition-Based Fluid Structure Interaction Algorithms for Highly Nonlinear and Anisotropic Elastic Arterial Wall Models in 3D[a]

[a] This project is supported by the Deutsche Forschungsgemeinschaft (DFG) and the Swiss National Fond (SNF) within a D-A-CH proposal; it is a common grant proposal by the PIs Axel Klawonn, Oliver Rheinbach, Alfio Quarteroni, Simone Deparis, Daniel Balzani, and Jörg Schröder.

Note that this envisaged usage scenario dictated the need for the development of a software library encapsulating `FEAP` as an assembly machine right from the start. Our solver `mparfeap` was in part developed as a proof of concept. This solver demonstrates both the feasibility of the technical approach, i.e., componentization of `FEAP`, and potential scalability of Newton-Krylov-FETI-DP for hyperelastic wall models. For the applications in FSI, the `FEAP` wrapper library `libfw-js` presented in Chapter 2 is used as a bridge component. It allows to use `FEAP` for the assembly in `LifeV`. More precisely, `FEAP-JS` assembles the quasi-static part in the fully time-dependent ALE-based fluid-structure interaction (FSI) simulations of arterial wall stresses in atherosclerotic arteries.

This invites us to take a look at a particular aspect of the aforementioned project from the point of view of software engineering:

Cross-Validation: The finite element discretizations for the St. Venant-Kirchhoff and Neo-Hooke material models available in both `FEAP` and `LifeV` have been compared, i.e., they have been shown to be consistent up to numerical tolerance [62]. This is an example for an independent *direct* cross-validation of new state of the art expression template-based implementations of the St. Venant-Kirchhoff and Neo-Hooke material models contributed to `LifeV` by P. Tricerri during his thesis work, see, e.g., Tricerri, Dedè, Quarteroni, and Sequeira [133]. Hence, this is a particularly reliable and secured basis regarding the correctnes of the modelling codes and in turn a sound basis for the development of authoritative massively parallel benchmark simulations in FSI. Cross-validation might be a valuabe quality assurance strategy also for other codes.

Boundary Conditions: Currently, only simple types of boundary conditions can be handled by `mparfeap`. The powerful features available to `FEAP` users for the realization of complex

loadings are for the most part intrinsically incompatible with our approach.[4] Thus, it would be an interesting addition to extend the `fddp` and `mparfeap` applications for the convenient handling of more complex and realistic boundary conditions for unstructured geometries in a massively parallel setting. Furthermore, problems in contact mechanics could be considered.

Let us now turn our attention towards future extensions regarding the application of linear and nonlinear domain decomposition methods in the context of nonlinear structural continuum mechanics. An important aspect of the mathematical formulation is the nonlinear, i.e., finite strain, strategy to incorporate the incompressibility constraint and how it interacts with FETI-DP. There are two major directions into which the present work could be extended.

Augmented Lagrange: In the present work we have only considered a simple penalty formulation in order to enforce quasi-incompressibility. This has been achieved. A possible improvement lies in the application Augmented Lagrange methods as presented in Brinkhues, Klawonn, Rheinbach, and Schröder [25] and in the dissertation of Brinkhues [24]. These could be integrated into `mparfeap`. This would allow to investigate these schemes for unstructured problems on the large scale, in particular with respect to their parallel scalability.

FETI-DP Coarse Spaces for Incompressible Linear Elasticity: In the context of the present work, we have used FETI-DP Algorithm C and Algorithm D_E. Both algorithms have been shown to be numerically scalable for linear elasticity *without incompressibility constraints* in three dimensions. Numerically scalable FETI-DP algorithms for *incompressible and almost incompressible* linear elasticity have previously been proposed and it would be interesting to study them in the context of finite strain incompressible hyperelasticity and in particular the effectiveness for soft biological tissue models.

Inexact FETI-DP Methods: We have only presented results for exact FETI-DP algorithms embedded into a Newton-Krylov-FETI-DP scheme. Exact FETI-DP algorithms are, in general, more stable but the memory requirements of direct factorization of the coarse problem is a limitaton. The next order of scalability requires appropriate iterative solution techniques for the coarse problem, i.e., inexact FETI-DP methods.

Nonlinear Domain Decomposition: A still recent development are nonlinear domain-decomposition based preconditioners acting on the level of the Newton scheme. Such approaches consequently lead to discontinuous Newton corrections. Nonlinear FETI-DP and BDDC algorithms have been investigated, e.g., by Klawonn, Lanser, and Rheinbach [86]; see also Klawonn, Lanser, Radtke, and Rheinbach [85]. Reduced communication costs are expected to yield algorithms scaling up to the exascale. For related work on applications of nonlinear domain decomposition methods in structural mechanics, see Gosselet and Rey [58, 59].

Adaptive Coarse Spaces: Another highly interesting modern approach to preconditioning is to adaptively compute optimal coarse spaces. This allows to improve the condition number of the preconditioned FETI-DP system in situations where not much is known about the operator a priori. Large reductions in the total number of FETI-DP iterations are possible. For recent work on adaptively computed coarse spaces in FETI-methods, see Klawonn, Radtke, and Rheinbach [78] and references therein; see also Klawonn, Lanser, Radtke, and Rheinbach [85].

Further, better analysis and control of the nonlinear dynamics of the solution curves φ_t considered as a function of the pseudo-time t during load stepping are of interest.

[4] We have abstained from usage of any `FEAP` functionality which manipulates the linear systems on the equation level.

Globalization: In the present work, we have essentially used load stepping as a globalization method. Other globalized Newton-schemes might also be useful to accelerate nonlinear solvers or increase their robustness. Note that conservative exterior loadings yield the structure of a minimization problem for a total energy of the mechanical system, i.e., leads to a symmetric Hessian. In general, however, one has to globalize the root finding problem arising from the contributions to the virtual work balance $G(u)$. Line search methods which reliably reduce work or increase stability in this context when used inside a load stepping scheme would certainly be of interest.

Stability Analysis: It would be interesting to extend `mparfeap` using strategies that allow to analyze nonlinear instabilities, e.g., buckling, and corresponding continuation methods. This is less immediate than it may seem at first glance as standard procedures require a robust eigenvalue analysis for the global tangent stiffness matrix, which can be extremely large and is never formed in our Newton-Krylov-FETI-DP approach.

A new door to unforeseen integrations fusing `FEAP` into other software projects is wide open now. New developments of nonlinear domain decomposition based techniques using well-established high-quality and high-performance implementations available in `FEAP` have thus been rendered possible and are already pursued.

In the hands of a creative and diligent mind a good tool may well show unanticipated worth. Only future can tell whether we have succeeded in providing such a tool; we do have strived for it; we do hope to be surprised . . . .

APPENDIX A

Hyperelasticity for Soft Biologial Tissues

Legal Disclaimer

This appendix is based on pages 20-41 of Section 2 Notation and Preliminaries of the present authors Diploma-thesis with the title

– The Planar Cosserat Model –
Minimization of the Shear Energy on SO(2) and Relations to Geometric Function Theory,

Technische Universität Darmstadt, 2007, advised by PD Dr. rer. nat. habil. Patrizio Neff.

In the following, we introduce the quasi-static setting of finite hyperelasticity in $\mathbb{R}^3$. We begin with some notation and introduce isotropic, anisotropic and incompressible structural models. For an extensive treatment of the subject based on abstract differential geometric principles, see Marsden and Hughes [97]. A treatment which is directed towards engineering sciences, see Holzapfel [63] and for a treatment written from the standpoint of functional analysis, we refer to the monograph by Ciarlet [29]. For a thorough introduction to nonlinear finite element methods for nonlinear continuum mechanics, see Wriggers [137].

A.1 Notation

$\mathbb{N}, \mathbb{Z}, \mathbb{R}$	Natural, integer and real numbers
$\mathrm{Hom}(V, W)$	Homomorphisms V to W
$\mathrm{M}(n, \mathbb{R})$	Square real $n \times n$-matrices
$\mathrm{Skew}(n, \mathbb{R}) \subset \mathrm{M}(n, \mathbb{R})$	Skew-symmetric: $\{A \in \mathrm{M}(n, \mathbb{R}) : A^T = -A\}$
$\mathrm{Sym}(n, \mathbb{R}) \subset \mathrm{M}(n, \mathbb{R})$	Symmetric: $\{S \in \mathrm{M}(n, \mathbb{R}) : S^T = S\}$
$\mathrm{PSym}(n, \mathbb{R}) \subset \mathrm{Sym}(n, \mathbb{R})$	Positive definite: $\{S \in \mathrm{Sym}(n, \mathbb{R}) : x \neq 0 \implies x^T S x > 0\}$
$\mathrm{GL}^+(n, \mathbb{R})$	Identity component of the general linear group
$\mathrm{SL}(n)$	Special linear group
$\mathrm{SE}(n)$	Special euclidean group (rigid body motions)
$\mathrm{SO}(n)$	Special orthogonal group
$\mathbb{0}_n \in \mathrm{M}(n, \mathbb{R})$	Zero matrix
$\mathbb{1}_n \in \mathrm{M}(n, \mathbb{R})$	Identity matrix
$C^\infty(M, N)$	Smooth maps between smooth manifolds mapping M to N
TM	Tangent bundle of a smooth manifold M
$d\varphi : TM \to TN$	Derivative of a smooth map $\varphi : M \to N$
$\nabla\varphi : \Omega \to \mathrm{M}(n, \mathbb{R})$	Matrix representation of the derivative
$\mathcal{A}$	Abstract space of admissible functions
$T\mathcal{A}$	Abstract space of variations of admissible functions
$\Omega = \mathrm{id}_\Omega(\Omega)$	Reference domain in configuration id_Ω
$\Omega_{\mathrm{def}} = \varphi(\Omega)$	Deformed domain in configuration φ
$\mathcal{T} = \{T_i\}_{1 \leq i \leq N_{\mathrm{elem}}}$	Tetrahedral triangulation of Ω

Definition A.1.1 (Projections)**.** *We denote the alternating, symmetric and deviatoric tensor projections for mixed second order tensors* $\mathrm{M}(3,\mathbb{R})$ *by*

$$\begin{aligned} \operatorname{skew} &: \mathrm{M}(3,\mathbb{R}) \to \operatorname{Skew}(3,\mathbb{R}), & \operatorname{skew}[X] &:= \frac{1}{2}(X - X^T) , \\ \operatorname{sym} &: \mathrm{M}(3,\mathbb{R}) \to \operatorname{Sym}(3,\mathbb{R}), & \operatorname{sym}[X] &:= \frac{1}{2}(X + X^T) , \\ \operatorname{dev} &: \mathrm{M}(3,\mathbb{R}) \to \mathit{ker}\operatorname{tr}[\cdot], & \operatorname{dev}[X] &:= X - \frac{1}{3}\operatorname{tr}[X]\cdot \mathbb{1} , \end{aligned}$$

respectively.

It is easy to see that $\operatorname{skew} + \operatorname{sym} = \mathrm{id}_{\mathrm{M}(3,\mathbb{R})}$ and $\operatorname{tr} \circ \operatorname{dev} = 0$. Moreover,

$$\mathrm{M}(3,\mathbb{R}) = \operatorname{Sym}(3,\mathbb{R}) \oplus_{\perp} \operatorname{Skew}(3,\mathbb{R}) , \tag{A.1.1}$$

where the orthogonality is relative to the usual matrix inner product

$$\langle X, Y \rangle := \operatorname{tr}\left[X^T Y\right] = \sum_{1 \le i,j \le n} X_{ij} Y_{ij} .$$

This scalar product induces a matrix norm.

Definition A.1.2 (Frobenius Matrix Norm)**.** *The* ***Frobenius norm*** *is given by*

$$\|X\|^2 := \langle X, X \rangle := \operatorname{tr}\left[X^T X\right] . \tag{A.1.2}$$

A.2 Finite Elasticity

Our focus is on simulations of structural mechanics in the elastic regime. The term *finite* is the antonym to *infinitesimal*. It hence implies *large deformations of solids* as opposed to the *infinitesimal* deformations of the reference configuration id_Ω which are the subject of the linearized theory, i.e., of classical linear elasticity. The terms *finite* and *nonlinear* are used interchangeably in the literature. For the introduction, we take an approach to the subject in terms of classical smooth functions. It is not our goal to prepare a setting to formulate an existence theory.

Definition A.2.1 (Body)**.** *A* ***body*** Ω *is a compact smooth three-dimensional manifold* $\Omega \subset \mathbb{R}^3$ *with piecewise smooth boundary* $\partial\Omega$*.*

For the application of finite element methods for the approximations of solutions to the arising boundary value problems, one typically considers domains Ω which are polyhedral with Lipschitz-boundary $\partial\Omega$.

Definition A.2.2 (Deformation Mapping and Deformed Body)**.** *An orientation preserving global* C^∞*-diffeomorphism*

$$\varphi : \Omega \to \Omega_{\mathrm{def}} \subset \mathbb{R}^3$$

shall be called a ***deformation*** *of the body* Ω *into the deformed body*

$$\Omega_{\mathrm{def}} := \varphi(\Omega) .$$

According to these definitions, the image $\varphi(\Omega)$ of a body is a body itself, because the global manifold structure is preserved by the deformation. Note that this choice does not allow a deformed body to intersect itself. The space of possible states of a mechanical system is commonly referred to as a configuration space. The configuration space of the elastic body Ω is the set of deformations.

Definition A.2.3 (Configuration Space of an Elastic Body)**.** *Let $\Omega \subset \mathbb{R}^3$ be a body. We define the configuration space of an elastic body as the set of global orientation preserving C^∞-diffeomorphisms*

$$\mathcal{C}_\Omega^{\text{elas}} := \{\varphi : \Omega \to \varphi(\Omega) : \varphi \text{ is a global diffeomorphism and } \det[\nabla\varphi] > 0\} .$$

This choice of configuration space excludes the possibility of local and global topology changes, because global diffeomorphisms are global and local homeomorphisms. This is intuitive for purely elastic processes. For the introduction we shall assume that configurations are classical smooth mappings, which does not lead to a sequentially complete space. The generalization to weakened notions of functions, i.e., to Sobolev functions and spaces, is hence essential for an existence theory. See, e.g., the elegant and rigorous exposition due to Ciarlet [29].

Definition A.2.4 (Elastic Body)**.** *We call a pair $(\Omega, \mathcal{C}_\Omega^{\text{elas}})$ of a body Ω and configuration space $\mathcal{C}_\Omega^{\text{elas}}$ an **elastic body**.*

Definition A.2.5 (Displacement Field)**.** *To any configuration $\varphi : \Omega \to \Omega_{\text{def}} \subset \mathbb{R}^3$ we can associate the **displacement field***

$$u : \Omega \to \mathbb{R}^3, \quad u(x) := \varphi(x) - \text{id}_\Omega(x) .$$

Note that $u = 0$ is equivalent to $\varphi = \text{id}_\Omega$. The deformation can be computed from the displacement field $\varphi(x) = \text{id}_\Omega(x) + u(x)$ and vice versa. One can take both perspectives.

Definition A.2.6 (Reference Configuration of a Body)**.** *Let Ω be an elastic body, i.e., a body with configuration space $\mathcal{C}_\Omega^{\text{elas}}$. The distinguished undeformed state of the body Ω is represented by the identity mapping*

$$\text{id}_\Omega \in \mathcal{C}_\Omega^{\text{elas}}, \quad \text{id}_\Omega(x) := x , \tag{A.2.3}$$

*Thus, id_Ω is called the **reference configuration**.*

By SE(3) we denote the **special Euclidean group** of isometries of $\mathbb{R}^3$. We shall also refer to it as the group of **rigid body motions**, since all internal distances in the body are left invariant by the corresponding group action.

Remark A.2.7 (Rigid Body)**.** *A body Ω with configuration space* SE(3)*, i.e.,*

$$\mathcal{C}_\Omega^{\text{rigid}} := \text{SE}(3) \tag{A.2.4}$$

*is called a **rigid body**.*

Noting the relation $\text{SE}(3) \subset \mathcal{C}_\Omega^{\text{elas}}$, we see that the configuration space of the rigid body is contained in the larger configuration space of the elastic body.

A first order theory depends on the derivative of the field variable. In the context of elasticity, it is customary to define the following.

Definition A.2.8 (The Deformation Gradient Field)**.** *Let $\varphi \in \mathcal{C}_\Omega^{\text{elas}}$ be a deformation of Ω. The matrix representation $\nabla\varphi : \Omega \to \text{GL}^+(3, \mathbb{R})$ of the derivative map $d\varphi : T\Omega \to T\Omega_{\text{def}}$ is called the **deformation gradient field** of φ and we denote it by*

$$\mathbf{F} := \nabla\varphi \in \text{GL}^+(3, \mathbb{R}) .$$

Remark A.2.9 (On the Type of F)**.** *The deformation gradient tensor is neither a gradient nor a simple tensor field. To be precise, $\varphi : \Omega \to \Omega_{\text{def}}$ induces a derivative map between the vector bundles $T\Omega$ and $T\Omega_{\text{def}}$. Marsden refers to such objects as two-point tensor fields in Marsden and Hughes [97]. This is natural, since F transforms tensorially under changes of coordinate systems in the reference and the deformed body. Another way of phrasing this is that the derivative $d\varphi$ transforms as a vector bundle-valued differential one-form taking values in the tangent bundle of the deformed configuration $T\Omega_{\text{def}}$, see, e.g. Kanso, Arroyo, Tong, Yavari, Marsden, and Desbrun [66]. A gradient is an object $\nabla f := df^\sharp$ obtained as the dual of a derivative one-form via a Riemannian metric g.*

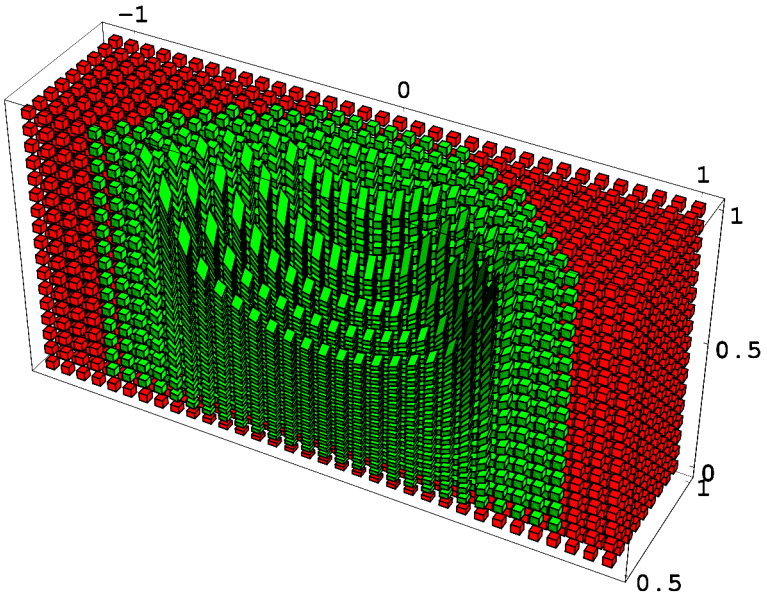


Figure A.2.1: The deformation gradient field $F_{\text{ind}} : T\Omega \to T\Omega_{\text{def}}$, $F_{\text{ind}} := \nabla\varphi_{\text{ind}}$, induced by an indented configuration $\varphi_{\text{ind}} : \Omega \to \Omega_{\text{def}}$ of the cuboid Ω. The deformed body is visualized by the action of F_{ind} on a grid of regularly distributed small cubes. Only the green part of the domain has been deformed.

The action of φ on infinitesimal cubes, i.e., on cubes in the tangent space $T_x\Omega$, is displayed by applying the linearization F. This is depicted in Figure A.2.1 for a synthetic example.

Definition A.2.10 (Stretch Tensors)**.** *Let $\varphi \in \mathcal{C}^{\text{elas}}_\Omega$ with deformation gradient $F \in \mathrm{GL}^+(3,\mathbb{R})$. We introduce the **right Cauchy-Green stretch tensor** $\mathbf{C}$ and the **right stretch tensor** or **Biot-stretch tensor** $\mathbf{U}$ as follows:*

$$\begin{aligned} C &: \mathrm{GL}^+(3,\mathbb{R}) \to \mathrm{PSym}(3,\mathbb{R}), \quad C(F) = F^T F\,, \\ U &: \mathrm{GL}^+(3,\mathbb{R}) \to \mathrm{PSym}(3,\mathbb{R}), \quad U(F) = \sqrt{C(F)} = \sqrt{F^T F}\,. \end{aligned} \tag{A.2.5}$$

Using the **right polar decomposition** given by $F = RU$, where $R \in \mathrm{SO}(3)$ and $U \in \mathrm{PSym}(3,\mathbb{R})$, we can refine this intuition by decomposing this infinitesimal deformation into a stretch U along the principal axes followed by a rotation R. This is illustrated in Figure A.2.2 for the case of planar simple shear.

Due to certain group invariances it is always possible to characterize the material response of isotropic materials in terms of three classical isotropic invariants I_1, I_2 and I_3 which depend on the right Cauchy-Green stretch tensor C. To this end, we introduce the following:

Definition A.2.11 (Principal Stretches)**.** *The three singular values of F ordered by descending magnitude*

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 > 0$$

are called the ***principal stretches*** *of U (or F, equivalently).*

This allows to introduce the classical principal invariants, which are given by the symmetric polynomials in the squared principial stretches of F, i.e., in the eigenvalues of C.

Definition A.2.12 (The Classical Principal Invariants of Isotropic Elasticity)**.** *The* ***principal invariants*** *are given by*

$$\begin{aligned} I_1 &:= \operatorname{tr}[C] = \sigma_1^2 + \sigma_2^2 + \sigma_3^2, \\ I_2 &:= \operatorname{tr}[\operatorname{cof}[C]] = \sigma_1^2\,\sigma_2^2 + \sigma_2^2\,\sigma_3^2 + \sigma_1^2\,\sigma_3^2, \\ I_3 &:= \det[C] = \sigma_1^2\,\sigma_2^2\,\sigma_3^2. \end{aligned}$$

Here, $\operatorname{cof}[C]$ *denotes the matrix of cofactors of C.*

The following example gives a geometric interpretation of the action of the deformation gradient F using the polar decomposition $F = RU$ into the associated right stretch tensor U and the right polar rotation R.

Example A.2.13 (Polar Decomposition of 2D Simple Shear)**.** *A simple shear with amount $\gamma \in \mathbb{R}$ of a planar body is given by*

$$F_\gamma = \begin{pmatrix} 1 & \gamma \\ 0 & 1 \end{pmatrix} \in \mathrm{SL}(2,\mathbb{R}), \tag{A.2.6}$$

acting via left multiplication on the reference configuration of a body. The singular values of F_γ are the square roots of the eigenvalues $\lambda_{1/2}$ of

$$C(F_\gamma) = F_\gamma^T F_\gamma = \begin{pmatrix} 1 & \gamma \\ \gamma & 1+\gamma^2 \end{pmatrix} \tag{A.2.7}$$

and we obtain

$$\sigma_{1/2} = \sqrt{\lambda_{1/2}} = \sqrt{\frac{1}{2}(2+\gamma^2 \pm \gamma\sqrt{4+\gamma^2})}\,.$$

An illustration is given in Figure A.2.2.

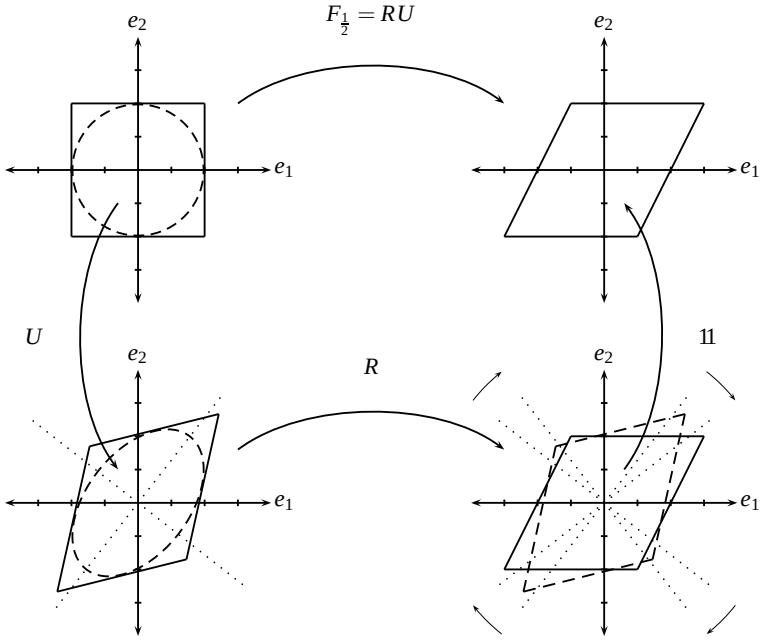


Figure A.2.2: The right polar decomposition decomposes the simple shear in e_1-direction with parameter $\gamma = \frac{1}{2}$ into a pure stretch U followed by a rotation R. The dotted lines show the principal directions of stretch, the stretch ellipse shows how the unit circle is deformed and the circular arrows illustrate the action of the polar rotation finally aligning the deformed shapes.

A.3 Quasi-static Isotropic Hyperelasticity

In quasi-static hyperelasticity, one assumes existence of a total energy functional for the interior forces induced by a deformation mapping contained in the space of admissible configurations. We shall denote this potential by

$$\Pi^{\text{int}} \ : \ \mathcal{A} \subseteq \mathcal{C}^{\text{elas}}_{\Omega} \to \mathbb{R}^{+}_{0} \ .$$

The term quasi-static refers to the assumption, that the time-dependent (transient) part of the equations of movement can be neglected. In the arising static theory, the functional Π^{int} plays the role of a stored energy potential and its first derivative corresponds to the internally generated force density due to the strain. We define an interior virtual work functional given by

$$G^{\text{int}} : T\mathcal{A} \to \mathbb{R}$$
$$G^{\text{int}}\big|_{\varphi}(\chi) := \left.\frac{d}{d\varepsilon}\right|_{\varepsilon=0} \Pi^{\text{int}}(\varphi + \varepsilon\chi) \ .$$

For a classical local first order single field theory it is assumed that the total energy can be expressed as an integral over a local energy density. The local strain energy density Ψ plays the role of the constitutive law and characterizes the mechanical response behavior of the material of the body. Such a theory postulates that the energy depends on the field only through the position $x \in \Omega$

and the first derivative of the field, i.e., on the **deformation gradient field** $F := \nabla\varphi$.[1] We thus introduce:

Definition A.3.1 (Local Strain Energy Density)**.** *We call a function*

$$\Psi : \Omega \times \mathrm{GL}^+(3,\mathbb{R}) \to \mathbb{R}_0^+$$

a ***local strain energy density*** $\Psi(x, F(x))$*. It is called homogeneous, if the dependence on* $x \in \Omega$ *is only through the deformation gradient* $F(x)$*. Thus, a homogeneous strain energy density can be written as* $\Psi(F)$*.*

For simplicity of exposition, we shall restrict our attention to the case of **homogeneous** strain energy densities $\Psi(F)$ in what follows.

Definition A.3.2 (Hyperelastic Body)**.** *A* ***hyperelastic body*** $(\Omega, \mathcal{C}_\Omega^{\mathrm{elas}}, \Psi)$ *is an elastic body* $(\Omega, \mathcal{C}_\Omega^{\mathrm{elas}})$ *together with a local strain energy density* $\Psi : \Omega \times \mathrm{GL}^+(3,\mathbb{R}) \to \mathbb{R}_0^+$*.*

A hyperelastic body allows to measure the internally stored strain energy induced by a given configuration by integration of the local strain energy density over the elastic body.

Definition A.3.3 (Internal Strain Energy)**.** *The* ***internal strain energy*** *of a hyperelastic body* Ω *in an* ***admissible configuration*** $\varphi \in \mathcal{A} \subset \mathcal{C}_\Omega^{\mathrm{elas}}$ *is given by the integral*

$$\Pi^{\mathrm{int}} : \mathcal{A} \to \mathbb{R}_0^+, \qquad \Pi^{\mathrm{int}}(\varphi) = \int_\Omega \Psi(F)\ dV\,. \tag{A.3.8}$$

Here, $\Psi : \mathrm{GL}^+(3,\mathbb{R}) \to \mathbb{R}_0^+$ *is a strain energy density and* $F := \nabla\varphi$ *is the deformation gradient of the configuration* φ*.*

Dirichlet boundary conditions of place are formulated by an appropriate choice of admissible configurations $\mathcal{A}$, satisfying these boundary conditions. Regarding other boundary conditions imposed in our applications, we consider the following formulation for the exterior contributions. To this end, we denote by $\partial\Omega_{\mathrm{def}_N} \subseteq \partial\Omega_{\mathrm{def}}$ the Neumann-boundary in the deformed configuration. A traction field $t_N : \partial\Omega_{\mathrm{def}_N} \to \mathbb{R}^3$ is to be imposed there. Further, let $\rho_0 : \Omega \to \mathbb{R}^+$ denote the physical density of the body and let $g : \mathbb{R}^3 \to \mathbb{R}^3$ be a volume force density.

Definition A.3.4 (Virtual Work due to Exterior Loadings)**.** *We define the* ***external virtual work*** *functional as*

$$\begin{aligned} G^{\mathrm{ext}} :&\quad T\mathcal{A} \to \mathbb{R}, \\ G^{\mathrm{ext}}\big|_\varphi(\chi) &= \underbrace{\int_{\partial\Omega_{\mathrm{def}_N}} \langle t_N,\, \chi \circ \varphi^{-1} \rangle\ dA}_{\textit{Pressure load}} + \underbrace{\int_\Omega \rho_0 \langle g,\, \chi \rangle\ dV}_{\textit{Volume force}}\,. \end{aligned} \tag{A.3.9}$$

In order to approximate the effects due to the blood pressure inside the arterial wall structure, a pressure load is imposed on the fluid-structure interface. We denote this part of the interface by $\partial\Omega_{\mathrm{def}_N}$ and consider a pressure induced traction field which is normal to the fluid-structure interface

$$t_N : \partial\Omega_{\mathrm{def}_N} \to \mathbb{R}^3, \qquad t_N := -\overline{p}\, n\,. \tag{A.3.10}$$

[1] Note that there are also multi-field theories. E.g., to model incompressibility, it is common to use a three-field Hu-Washizu approach. Cosserat and micromorphic models with rotational degrees of freedom and micromorphic material models are other noteworthy examples, see e.g. Neff [100] and Neff and Forest [101] and references therein.

The pressure is assumed to be constant $\overline{p} \geq 0$ [kPa]. Note that the reflected outward unit normal field given by $-n$ determines an inward pointing normal field. Thus, in our applications, a positive blood pressure causes the arterial wall structure to expand. Pressure loadings are a non-trivial type of boundary condition. Let us illuminate some consequences of such boundary conditions:

Remark A.3.5 (On Configuration-Dependent Pressure Loads)**.** *Pressure loadings are configuration dependent. Such loads are also referred to as "follower loads". The virtual work due to such loads can in general not be derived from a potential function. This is in strong contrast to the case of dead loads and the effect is not related to dissipation. In general, the first variation of the functional of the virtual work* G^{ext} *is non-symmetric which implies a possibly* non-symmetric *finite element stiffness matrix* $K(\varphi^h)$*. This is well-known (see, e.g.,Wriggers [137]), however, the symmetry depends in a non-trivial way on the boundary conditions; cf. Romano [111] and Sewell [120]. For the linearization and implementation of the pressure loadings used in our applications, we refer to Simo, Taylor, and Wriggers [123]. For the existence of a potential for the surface tractions due to constant pressure loading and for conditions implying symmetry, see Romano [111]. Further, note the extensive work due to Bufler and Schweizerhof of which we only want to mention Bufler [26] and Rumpel and Schweizerhof [112] here. In practice the contribution to the stiffness matrix due to the pressure loading is often "nearly" symmetric. Consequently, we consider a symmetric approximation* $K^S(\varphi^h) \approx K(\varphi^h)$*. This implies that we consider a* possibly inconsistent *tangent for the sake of memory conservation. The latter is advantageous for parallel scalability.*

Given the previously introduced setup, the problem is to compute an admissible configuration in equilibrium.

Equilibrium of Forces

A hyperelastic body $(\Omega, \mathcal{C}^{\text{elas}}_{\Omega}, \Psi)$ is in equilibrium in a given admissible configuration $\varphi \in \mathcal{A} \subseteq \mathcal{C}^{\text{elas}}_{\Omega}$, if and only if

$$\forall \chi \in T_{\varphi}\mathcal{A} \quad : \quad G^{\text{int}}|_{\varphi}(\chi) \quad - \quad G^{\text{ext}}|_{\varphi}(\chi) = 0 \, . \tag{A.3.11}$$

We shall refer to contributions to the exterior work functional which derive from a potential as **conservative** loads and denote the corresponding exterior load potential by

$$\Pi^{\text{ext}}_{(\text{cons})} \; : \; \mathcal{A} \subseteq \mathcal{C}^{\text{elas}}_{\Omega} \to \mathbb{R}^{+}_{0} \; .$$

Further, we introduce a functional which measures the difference of the internal and external potentials

$$\Pi(\varphi) \quad := \quad \Pi^{\text{int}}(\varphi) \quad - \quad \Pi^{\text{ext}}_{(\text{cons})}(\varphi) \, .$$

In our given setting, it is only the contribution due to the volume forces in G^{ext} which is guaranteed to derive from a potential. So-called dead loads, i.e., surface traction loads which do not follow the configuration can also be described by a contribution to the potential $\Pi^{\text{ext}}_{(\text{cons})}$.

There are a few fundamental invariance conditions for local strain energy densities and we start the presentation with a definition.

Definition A.3.6 (Objectivity and Isotropy)**.** *Let* $\Psi : \mathrm{GL}^{+}(3, \mathbb{R}) \to \mathbb{R}^{+}_{0}$ *denote a local strain energy density, then* Ψ *is called*

i.) ***Objective:*** $\forall Q \in \mathrm{SO}(3) : \quad \Psi(QF) = \Psi(F) \; ,$

ii.) ***Isotropic:*** $\forall Q \in \mathrm{SO}(3) : \quad \Psi(FQ) = \Psi(F) \; ,$

if and only if $\Psi(F)$ satisfies the respective condition on the right hand side.

The previously introduced notion of objectivity is also referred to as **frame indifference**. It is a necessary requirement for any free energy density Ψ. A strain energy density Ψ which is frame indifferent and isotropic satisfies the relation

$$\forall Q_1, Q_2 \in \mathrm{SO}(3): \quad \Psi(Q_1 F Q_2) = \Psi(F)$$

which is called **orthogonal invariance**.

Orthogonally invariant strain energies can always be cast into a particular form depending only on the classical principal invariants.

Lemma A.3.7 (Representation of Isotropic Strain Energies)**.** *An isotropic strain energy density can always be considered as a function of C or, equivalently, as a function of the principal invariants of C, i.e., there exist representations*

$$\Psi(F) = \widehat{\Psi}(C) = \widehat{\widehat{\Psi}}(I_1(C),\, I_2(C),\, I_3(C)) \,.$$

Proof. Standard. □

Let us introduce some well-known examples for isotropic strain energies.

Example A.3.8 (Some Classical Strain Energies)**.** *The classical Biot, St. Venant-Kirchoff and Hencky strain energy measures are given by*

$$\begin{aligned}
\Psi_{\mathrm{Biot}}(F) &= \mu \, \|U(F) - \mathbb{1}\|^2 + \frac{\lambda}{2} \, \mathrm{tr}\,[U(F) - \mathbb{1}]^2, \\
\Psi_{\mathrm{STVK}}(F) &= \frac{\mu}{4} \, \|C(F) - \mathbb{1}\|^2 + \frac{\lambda}{8} \, \mathrm{tr}\,[C(F) - \mathbb{1}]^2, \\
\Psi_{\mathrm{Hencky}}(F) &= \mu \, \|\log(U(F))\|^2 + \frac{\lambda}{2} \, \mathrm{tr}\,[\log(U(F))]^2 \,.
\end{aligned} \tag{A.3.12}$$

The parameters λ and μ are called the first and second Lamé-parameters, respectively, and their associated SI-unit is the pascal [Pa] := $[\frac{\mathrm{N}}{\mathrm{m}^2}]$ which is force per square meter and hence a pressure. The second parameter μ is commonly referred to as the shear modulus.

Let us assume that we may apply Taylor's theorem to the internal strain energy in the reference configuration. Considering the displacement u as a small perturbation of $\varphi = \mathrm{id}_\Omega + u$, we formally write

$$\Pi^{\mathrm{int}}(\mathrm{id}_\Omega + u) = \Pi^{\mathrm{int}}(\mathrm{id}_\Omega) \; + \; d\Pi^{\mathrm{int}}_\varphi(\varphi)|_{\mathrm{id}_\Omega}(u) \; + \; \frac{1}{2}\, d^2_\varphi \Pi^{\mathrm{int}}(\varphi)|_{\mathrm{id}_\Omega}(u,u) \; + \; \ldots \tag{A.3.13}$$

$$= \int_\Omega \underbrace{\Psi(\mathbb{1})}_{=0} \; + \; \underbrace{d_F \Psi(F)|_{\mathbb{1}}(\nabla u)}_{\langle S_1(\mathbb{1}),\, \nabla u \rangle = 0} + \frac{1}{2}\, d^2_F \Psi(F)|_{\mathbb{1}}(\nabla u, \nabla u) + \; \ldots \; \mathrm{dV} \,. \tag{A.3.14}$$

Definition A.3.9 (Linearization Equivalence and Linear Elasticity)**.** *We call a local strain energy density $\Psi(F)$ **linearization equivalent** to linear elasticity if the truncated Taylor expansion of $\Pi^{\mathrm{int}}(\mathrm{id}_\Omega + u)$ up to the quadratic terms in u leads to a potential for the strain energy in linear elasticity.*

Linearization equivalence of a finite hyperelastic formulation with classical linear elasticity is generally considered a basic requirement for any strain energy density $\Psi(F)$. To give an example,

we shall now verify the consistency of the St. Venant-Kirchhoff energy with linear elasticity in the isotropic case. To this end, we note that

$$C(\mathbb{1}+\nabla u)-\mathbb{1} = (\mathbb{1}+\nabla u)^T(\mathbb{1}+\nabla u)-\mathbb{1} \tag{A.3.15}$$
$$= \mathbb{1}+\nabla u+(\nabla u)^T+(\nabla u)^T\nabla u-\mathbb{1} \tag{A.3.16}$$
$$= 2\,\mathrm{sym}\,[\nabla u]+(\nabla u)^T\nabla u\,. \tag{A.3.17}$$

Example A.3.10 (Linearization Equivalence of the St. Venant-Kirchhoff with Linear Elasticity)**.** *Expanding the strain energy Ψ_{STVK} with respect to the perturbation u in id_Ω, we obtain the linearization of the St. Venant-Kirchhoff strain energy density as follows:*

$$\Psi_{\mathrm{STVK}}(\mathbb{1}+\nabla u) = \frac{\mu}{4}\,\|C(\mathbb{1}+\nabla u)-\mathbb{1}\|^2+\frac{\lambda}{8}\,\mathrm{tr}\,[C(\mathbb{1}+\nabla u)-\mathbb{1}]^2 \tag{A.3.18}$$
$$= \frac{\mu}{4}\,\left\|2\,\mathrm{sym}\,[\nabla u]+(\nabla u)^T(\nabla u)\right\|^2+\frac{\lambda}{8}\,\mathrm{tr}\left[2\,\mathrm{sym}\,[\nabla u]+(\nabla u)^T(\nabla u)\right]^2 \tag{A.3.19}$$
$$= \mu\,\|\mathrm{sym}\,[\nabla u]\|^2+\frac{\lambda}{2}\,\mathrm{tr}\,[\nabla u]^2\;+\;\dots\,. \tag{A.3.20}$$

Dropping the suppressed higher order terms, we obtain the well-known potential for linear elasticity. This shows the claim.

The notion of a strain energy density immediately leads to the concept of an associated stress in the material which is induced from a given deformation and encoded in a stress tensor field with associated SI-unit [Pa].[2] In hyperelasticity the stress tensor fields are obtained as suitable derivatives of the strain energy density. This corresponds to the principle that conservative forces arise as derivatives of potentials. To write the stress tensors in matrix form, we introduce the representation of derivatives of stored energy densities in the matrix inner product

$$\forall X\in \mathrm{M}(3,\mathbb{R}):\quad d_F\Psi|_{F_0}(X)\;=\;\left\langle D_F\Psi|_{F_0},\,X\right\rangle.$$

We can now introduce the classical stress tensors in the context of hyperelasticity.[3]

Definition A.3.11 (Piola and Cauchy Stress Tensors)**.** *Let $\varphi\in\mathcal{C}^{\mathrm{elas}}_\Omega$ be a deformation with associated deformation gradient field $F:=\nabla\varphi$. The* ***first and second Piola-Kirchhoff stress tensors*** *S_1, S_2 and the* ***Cauchy stress tensor*** *σ are given by*

$$S_1(F):=D_F\Psi(F),\quad S_2(C):=2\,D_C\,\widehat{\Psi}(C)=F^{-1}S_1(F)\quad,\text{and} \tag{A.3.21}$$
$$\sigma(F):=\frac{1}{\det[F]}S_1(F)F^T=\frac{1}{\det[F]}FS_2(C(F))F^T\,. \tag{A.3.22}$$

The relation between the first and second Piola-Kirchhoff stress tensors is also called the Doyle-Erickson formula.

Lemma A.3.12. *The Doyle-Ericksen formula is consistent with the usual transformation rule*

$$S_1=FS_2\,.$$

[2] Due to the choice of material parameters given in [kPa], we however consider the induced unit [kPa] in the present work.

[3] In Kanso, Arroyo, Tong, Yavari, Marsden, and Desbrun [66] the Cauchy stress is more geometrically characterized as a covector-valued two-form. Such an object maps an infinitesimal surface element in the deformed configuration to the traction force it is subjected to.

Proof. Applying the chain rule, we immediately obtain that

$$\begin{aligned}
\big\langle S_1(F_0),\, X\big\rangle &= d_F\Psi(F)|_{F_0}(X)\\
&= d_F\,\widehat{\Psi}(C(F))|_{F_0}(X)\\
&= d_C\,\widehat{\Psi}\,|_{C_0}\circ d_FC|_{F_0}(X)\\
&= \big\langle \frac{1}{2}S_2(C_0),\, d_FC|_{F_0}(X)\big\rangle\\
&= \big\langle \frac{1}{2}S_2(C_0),\, 2\,\mathrm{sym}\left[F_0^TX\right]\big\rangle\\
&= \big\langle S_2(C_0),\, \mathrm{sym}\left[F_0^TX\right]\big\rangle + \underbrace{\big\langle S_2(C_0),\, \mathrm{skew}\left[F_0^TX\right]\big\rangle}_{=\,0}\\
&= \big\langle S_2(C_0),\, F_0^TX\big\rangle\\
&= \big\langle F_0S_2(C_0),\, X\big\rangle.
\end{aligned}$$

The equality holds for all pairs (F_0, X) which establishes the general relation $S_1(F) = F\,S_2(C(F))$ in the context of hyperelasticity.[4] □

In the following *we shall always assume that the body is stress-free in the reference configuration* id_Ω. This holds also for the simulations presented in Chapter 3 where the pre-stresses, also called eigenstresses or residual stresses, in the arterial walls have not been modeled. In the case of no eigenstresses, we have that

$$\sigma(\mathbb{1}) = S_1(\mathbb{1}) = S_2(\mathbb{1}) = 0\,.$$

We note that the condition $\Psi(\mathbb{1}) = 0$ implies that $d_F\Psi(F)|_{\mathbb{1}} = S_1(F) = 0$ since $\Psi(F) \geq 0$. Thus, if a strain energy density satisfies the previous condition, the reference configuration is necessarily stress-free.

Lemma A.3.13 (Objectivity and Symmetry of Cauchy Stresses)**.** *Suppose that* $\Psi : \mathrm{GL}^+(3,\mathbb{R}) \to \mathbb{R}_0^+$ *is a differentiable, frame indifferent free energy, i.e.,*

$$\forall Q \in \mathrm{SO}(3): \quad \Psi(QF) = \Psi(F)\,.$$

Then, the Cauchy stress tensor σ *is necessarily symmetric.*

The proof illustrates the interaction of the group invariance with the differentiability of Ψ.

Proof. Let $Q : (-\varepsilon, \varepsilon) \to \mathrm{SO}(3)$ be a smooth curve in the matrix Lie-group $\mathrm{SO}(3)$, with $Q(0) = \mathbb{1}$. Then $\dot{Q}(0) = A \in \mathfrak{so}(3) \cong \mathrm{Skew}(3,\mathbb{R})$ and, due to the objectivity of Ψ, we obtain

$$\begin{aligned}
0 = \left.\frac{d\Psi(Q(t)F)}{dt}\right|_{t=0} &= d_F\Psi|_{Q(t)F|_{t=0}} \circ \left.\frac{dQ(t)F}{dt}\right|_{t=0} = d_F\Psi|_F\,(\dot{Q}(0)F)\\
= d_F\Psi|_F\,(\mathbb{1}\,AF) &= \big\langle S_1(F),\, AF\big\rangle = \big\langle S_1(F)\,F^T,\, A\big\rangle = \big\langle \det[F]\sigma(F),\, A\big\rangle\,.
\end{aligned}$$

Since A is skew-symmetric and $\det[F] > 0$, it follows that $\big\langle \mathrm{skew}\,[\sigma(F)]\,,\, A\big\rangle = 0$. Now, choosing

$$A = \mathrm{skew}\,[\sigma(F)] \quad\Longrightarrow\quad \|\mathrm{skew}\,[\sigma(F)]\|^2 = 0 \quad\Longrightarrow\quad \sigma(F) \in \mathrm{Sym}(3,\mathbb{R})$$

and hence the claim. □

[4] Here, we have inserted the derivative

$$d_FC|_{F_0}(X) = \left.\frac{d}{dt}\right|_{t=0} C(F_0 + tX) = 2\,\mathrm{sym}\left[F_0^TX\right].$$

Thus, in hyperelastic frame-indifferent elasticity theories with *simple* materials the Cauchy stress tensor σ, formally defined as in (A.3.22) is *always* symmetric.[5] We note also that S_1 is not symmetric in general and that the symmetry of S_2 immediately follows from the symmetry of the Cauchy stress tensor.

A.4 Anisotropy and Mixed Invariants

We recall that a strain energy density Ψ is called isotropic, if and only if it is invariant with respect to the right action of $\mathrm{SO}(3)$ on $\mathrm{GL}^+(3,\mathbb{R})$, i.e., it satisfies

$$\forall Q \in \mathrm{SO}(3): \quad \Psi(FQ) = \Psi(F) \ .$$

This can be interpreted as a perfect spherical symmetry reflecting that the material produces the same energetic response in every direction. If this is not the case, the material is said to be anisotropic. Typical anisotropic materials are crystals which show a material behavior which is strongly influenced by the symmetry group of the underlying crystal lattice. Another typical example is *transversal isotropy* which is characterized by a symmetry axis called the *preferred direction*. Fiber reinforced materials are a standard example. In the context of soft biological tissues the preferred direction is naturally given by muscle fibers.

A symmetry is characterized by an invariance with respect to a group action. For the action of a group $\mathcal{G}$ on an object X, we write symbolically $\mathcal{G} \curvearrowright X$. Further, we denote the corresponding group orbit of X by

$$\mathrm{Orb}_{\mathcal{G}}(X) := \{Q \curvearrowright X : Q \in \mathcal{G}\} \ .$$

Definition A.4.1 (Anisotropic Strain Energy Density)**.** *Let $\Psi(F)$ be a strain energy density. If*

$$\exists \widehat{Q} \in \mathrm{SO}(3): \quad \Psi(F\widehat{Q}) \neq \Psi(F) \ , \tag{A.4.23}$$

then $\Psi(F)$ is called an ***anisotropic strain energy density****.*

Definition A.4.2 (Material Symmetry Group)**.** *Let Ψ be an anisotropic free energy. The maximal subgroup $\mathcal{G} < \mathrm{SO}(3)$ satisfying*

$$\forall Q \in \mathcal{G}: \quad \Psi(FQ) = \Psi(F) \tag{A.4.24}$$

is called the ***material symmetry group*** *$\mathcal{G}$ of $\Psi(F)$.*

In order to construct invariants for anistropic energy densities the following concept is helpful.

Definition A.4.3 (Structural Tensor)**.** *Let $\mathcal{G}$ be a material symmetry group. A tensor T is called a* ***structural tensor*** *for a group action $\mathcal{G} \curvearrowright T$, if it is invariant under the group action*

$$\forall Q \in \mathcal{G}: \quad Q \curvearrowright T = T \ .$$

Structural tensors allow to express anisotropic stored energy densities as isotropic ones. Extending the argument list of the strain energy by a structural tensor, one finds that

$$\widehat{\Psi}(QCQ^T, QMQ^T) = \Psi(FQ^T, QMQ^T) = \Psi(F, M) \tag{A.4.25}$$

must hold. This is also called the principle of isotropy of space, cf. Ebbing [42]. Note that, in general, due to the anisotropy

$$\widehat{\Psi}(QCQ^T, M) = \Psi(FQ^T, M) \neq \Psi(F, M) \ . \tag{A.4.26}$$

[5] Note that in extended continuum theories, e.g., in Cosserat theory, the Cauchy stress tensor need not be symmetric, even if the theory models an isotropic material Neff, Fischle, and Münch [102] and Fischle, Neff, and Münch [51].

Example A.4.4 (Material Symmetry Group for Transversal Isotropy)**.** *Let $a \in T_x\Omega \cong \mathbb{R}^3$ denote a preferred direction. The material symmetry group for transversal isotropy is given by the stabilizer subgroup*

$$\mathcal{G}_{\text{ti}}(a) := \text{SO}(3)_a := \{Q \in \text{SO}(3) : Qa = a\} .$$

Example A.4.5 (A Structural Tensor for Tranversal Isotropy of 2nd-order)**.** *Let $a \in \mathbb{R}^3$ with $\|a\| = 1$ and introduce a matrix (mixed second order-tensor) through the dyadic product*

$$M(a) := a \otimes a := aa^T. \tag{A.4.27}$$

This is a structural tensor relative to the special orthogonal group $\text{SO}(3)$*, as we shall see now. Let us shortly investigate how the structural tensor is related to this group. Consider thus the left-action of the material symmetry group $\mathcal{G}_{\text{ti}}(a) = \text{SO}(3)_a$ on $\mathbb{R}^3$ through $L : \mathcal{G} \times \mathbb{R}^3 \to \mathbb{R}^3$, $L_Q(a) = Qa$. This directly induces a group action on $M(a)$, namely*

$$Q \curvearrowright M(a) := M(L_Q(a)) .$$

Now, since $Qa = a$, we find

$$\forall Q \in \mathcal{G}_{\text{ti}}(a) : \quad M(L_Q(a)) = M(Qa) = M(a) . \tag{A.4.28}$$

On the other hand

$$M(L_Q(a)) = (Qa)(Qa)^T = Q(aa^T)Q^T = QM(a)Q^T . \tag{A.4.29}$$

This implies the relation

$$\forall Q \in \mathcal{G}_{\text{ti}}(a) : \quad QM(a)Q^T = M(a)$$

which shows that the 2nd-order structural tensor $M(a)$ is related to the material symmetry group $\mathcal{G}_{\text{ti}}(a)$ as follows

$$\text{SO}(3)_{M(a)} = \mathcal{G}_{\text{ti}(a)} = \text{SO}(3)_a .$$

Clearly, the stabilizer on the left hand side is to be understood to be defined relative to the conjugate group action.

As in the isotropic case, there is a developed invariant theory for anisotropic material densities formulated in terms of structural tensors; see, e.g., Boehler [19]. Let us consider a preferred direction $a \in \mathbb{R}^3$ which satisfies $||a|| = 1$. Then, the matrix powers of the structural tensor are idempotent, i.e.,

$$M(a)^k = M(a), \quad \forall k \geq 2$$

Definition A.4.6 (The System of Basic Invariants for Isotropy)**.** *The **basic invariants** for isotropy are given by*

$$J_1 := \text{tr}\left[C^1\right], \quad J_2 := \text{tr}\left[C^2\right], \quad J_3 := \text{tr}\left[C^3\right] . \tag{A.4.30}$$

There is a one-to-one correspondence between the basic invariants J_k, $1 \leq k \leq 3$ and the principal invariants I_k, $1 \leq k \leq 3$ given by

$$J_1 = I_1, \qquad J_2 = I_1^2 - 2I_2, \qquad J_3 = I_1^3 - 3I_1I_2 + 3I_3 , \tag{A.4.31}$$

$$I_1 = J_1, \qquad 2\,I_2 = J_1^2 - J_2, \qquad 6\,I_3 = J_1^3 - 3J_1J_2 + 2J_3 . \tag{A.4.32}$$

The invariance of the basic invariants with respect to conjugation by SO(3) is easy to see. The conjugation is induced from the right action $R_Q(F) = FQ$ of $Q \in \mathrm{SO}(3)$ on F by

$$C(R_Q(F)) = (R_Q(F))^T (R_Q(F)) = (FQ)^T FQ = Q^T F^T FQ = Q^T C Q \, .$$

Definition A.4.7 (Mixed Invariants for Transversal Anisotropy)**.** *Let $a \in \mathbb{R}^3$ denote a preferred direction, $\mathcal{G}_{\mathrm{ti}}(a)$ the corresponding transversely isotropic material symmetry group, and $M(a)$ an associated structural tensor for transversal anistropy. Then the associated* ***mixed (basic) invariants*** *for transversal isotropy are given by*

$$J_4 := \mathrm{tr}\,[CM] \quad \textit{and} \quad J_5 := \mathrm{tr}\,\big[C^2 M\big]. \tag{A.4.33}$$

There is an intuitive geometric interpretation of the mixed invariant J_4. Due to

$$\mathrm{tr}\,[CM] = \big\langle Ca,\, a\big\rangle = \|Fa\|^2 = \|RUa\|^2 = \|Ua\|^2 \ ,$$

it measures the quadratic length change of the preferred direction a induced by the deformation φ. In principle higher mixed invariants can be constructed. However, they are not independent if the preferred direction $a \in \mathbb{R}^3$ satisfies $\|a\| = 1$. This can be seen as follows

$$\begin{aligned} J_6 &:= \mathrm{tr}\,\big[CM^2\big] = \mathrm{tr}\,\big[M^2 C\big] = \mathrm{tr}\,[MC] = \mathrm{tr}\,[CM] = J_4 \ , \quad \text{and} \\ J_7 &:= \mathrm{tr}\,\big[C^2 M^2\big] = \mathrm{tr}\,\big[M^2 C^2\big] = \mathrm{tr}\,\big[MC^2\big] = \mathrm{tr}\,\big[C^2 M\big] = J_5 \ . \end{aligned}$$

The extension to the case of two superimposed preferred directions $a_1, a_2 \in \mathbb{R}^3$ is as follows. Using two associated structural tensors suitable for transversal isotropy given by

$$M^{(k)} := M(a_k) = a_k \otimes a_k := a_k a_k^T, \quad k = 1, 2 \ ,$$

we obtain associated mixed basic invariants

$$J_4^{(k)} = \mathrm{tr}\,\Big[CM^{(k)}\Big], \quad \text{and} \quad J_5^{(k)} = \mathrm{tr}\,\Big[C^2 M^{(k)}\Big], \quad k = 1, 2 \ .$$

These correspond to the associated transversely isotropic material symmetry groups $\mathcal{G}_{\mathrm{ti}}(a_k)$, for $k = 1, 2$. The generalization to an arbitrary number of preferred directions is straight-forward.

Lemma A.4.8 (Invariants for Two Superimposed Preferred Directions)**.** *Let $\|a_k\| = 1, k = 1, 2$ be two distinct preferred directions. The following set of independent invariants*

$$\mathcal{I}_{\mathrm{aniso}} = \{I_1,\ I_2,\ I_3,\ J_4^{(k)},\ J_5^{(k)}\} \ , \quad k = 1, 2 \ . \tag{A.4.34}$$

is complete.

We shall not give a rigorous definition of a complete system of polynomial invariants here. It shall suffice to say, that there are no other invariants which have to be considered. As a starting point on invariant theory with a focus on applications in the context of polyconvex anisotropic energy densities, we point the interested reader to the dissertation by Ebbing [42].

We follow the exposition in Balzani [10] and consider strain energy densities with two superimposed preferred directions of the following form

$$\widehat{\Psi}(C, M^{(1)}, M^{(2)}) := \widehat{\Psi}^{\mathrm{igs}}(I_1, I_2, I_3) + \sum_{k=1}^{2} \widehat{\Psi}_{\mathrm{ti}}^{(k)}(I_1, I_2, I_3, J_4^{(k)}, J_5^{(k)}) \ .$$

In our applications on biomechanics, the first part models an isotropic ground substance $\widehat{\Psi}^{\text{igs}}$ and the terms $\widehat{\Psi}_{\text{ti}}^{(k)}$ in the sum model the energetic response for each of the muscle fiber directions in an arterial wall layer. The two preferred directions are weakly coupled in the local strain energy density. For the simulations in the present work we shall only consider energy densities which are of this type. I.e., energies where each contribution is separately polyconvex, cf. Brands, Klawonn, Rheinbach, and Schröder [23] and the thesis by Balzani [10].

The material symmetry group for such a model with multiple preferred directions $a_1, a_2 \in \mathbb{R}^3$ is in fact trivial. To see this, we consider the material symmetry groups $\mathcal{G}_{\text{ti}}^{(1)}$ and $\mathcal{G}_{\text{ti}}^{(2)}$. Clearly, if a_1 and a_2 are linearly independent $\mathcal{G}_{\text{ti}}^{(1)}$ does not leave $M^{(2)}$ fixed and vice versa. Thus, the total strain energy $\widehat{\Psi}(C, M^{(1)}, M^{(2)})$ will, in general, only admit a trivial material symmetry group

$$\mathcal{G} = \mathrm{SO}(3) \cap \mathcal{G}_{\text{ti}}^{(1)} \cap \mathcal{G}_{\text{ti}}^{(2)} = \{\mathbb{1}\} .$$

Let us now turn our focus towards generalized convexity conditions in the calculus of variations.

A.5 Generalized Convexity Conditions

The development of the existence theory in finite elasticity due to Sir John Ball hinges crucially on a notion called polyconvexity; see Ball [8]). Polyconvexity is one of multiple generalized convexity conditions. The availability of an existence theory is clearly the main motivation for the use of polyconvex energy functions. For an introduction to generalized convexity conditions in the calculus of variations, see Dacorogna [31] and Schröder and Neff [115]. We shall only briefly illuminate the notion of polyconvexity here.

Definition A.5.1 (Polyconvex Strain Energy Density)**.** *A strain energy density function $\Psi(F)$ is called **polyconvex** if there exists a convex function $P : \mathrm{M}(3,\mathbb{R}) \times \mathrm{M}(3,\mathbb{R}) \times \mathbb{R} \to \mathbb{R}$ such that*

$$\forall F \in \mathrm{M}(3,\mathbb{R}) : \quad \Psi(F) = P(F, \operatorname{cof}[F], \det[F]) .$$

A closely related notion are the Legendre-Hadamard conditions. We introduce the following standard notation for rank-one matrices $\xi \otimes \eta := \xi\eta^T$, where $\xi, \eta \in \mathbb{R}^3$.

Definition A.5.2 (Legendre-Hadamard Conditions)**.** *A bilinear form $a : \mathrm{M}(3,\mathbb{R}) \times \mathrm{M}(3,\mathbb{R}) \to \mathbb{R}$ is said to satisfy*

1. *the **Legendre-Hadamard** condition, if*

$$\forall \xi, \eta \in \mathbb{R}^3 : \quad a(\xi \otimes \eta, \xi \otimes \eta) \geq 0 ,$$

2. *the **strict Legendre-Hadamard** condition, if*

$$\forall \xi, \eta \in \mathbb{R}^3 \setminus \{0\} : \quad a(\xi \otimes \eta, \xi \otimes \eta) > 0 ,$$

3. *the **uniform Legendre-Hadamard** condition, if*

$$\exists c^+ > 0,\ \forall \xi, \eta \in \mathbb{R}^3 : \quad a(\xi \otimes \eta, \xi \otimes \eta) \geq c^+ ||\xi||^2 ||\eta||^2 .$$

In the context of the calculus of variations, the following relation between generalized notions of convexity is well-known; cf. Schröder and Neff [115]:

$$\text{Convexity} \implies \text{Polyconvexity} \implies \text{Quasi-convexity} \implies \text{Rank-one convexity}. \tag{A.5.35}$$

The reverse implications are in general not satisfied. Note also that convexity of a strain energy density $\Psi(F)$ in F implies non-physical growth conditions. It is thus a too strict requirement. This is not the case for polyconvex energy densities on which we focus our attention. The notion of polyconvexity was adapted to the anisotropic case in [114] by construction of polyconvex anisotropic invariants.

A.6 Polyconvex Anisotropic Strain Energy Densities

There are composition rules for polyconvex terms which preserve polyconvexity; polyconvex terms may be additively super-imposed for example. Using available composition rules, it is possible to construct classes of polyconvex strain energy densities with specific properties, based on a complete system of *polyconvex* anisotropic invariants. The previously introduced system of invariants

$$\mathcal{I}_{\text{aniso}} = \{I_1,\ I_2,\ I_3,\ J_4^{(k)},\ J_5^{(k)}\}$$

is not satisfactory in this regard. It contains an invariant $J_5^{(k)}$ which is not polyconvex; see Merodio and Neff [98]. For the remedy, one may consider a modified structural tensor

$$D(a) := \mathbb{1} + M(a)$$

relative to the material symmetry group $\mathcal{G}_{\text{ti}}(a)$. This modification was used in Schröder and Neff [114] to construct a dependent but polyconvex invariant for a given preferred direction $a \in \mathbb{R}^3$ which is given by

$$K_3(C, M(a)) := \operatorname{tr}[\operatorname{cof}[C]D(a)] = \operatorname{tr}[\operatorname{cof}[C](1 - M(a))] = I_1 J_4 - J_5\ . \tag{A.6.36}$$

Also in Schröder and Neff [114], the relation

$$K_3(C(F), M(a)) = \|\operatorname{cof}[F]\|^2 - \|\operatorname{cof}[F]a\|^2$$

is derived. This allows to give K_3 an interpretation as a measure for the change of area in the plane perpendicular to the preferred direction a. This interpretation is motivated by Nanson's formula for the change of the surface area element. It it easy to see that the invariants J_5 and K_3 are polynomially interdependent

$$K_3 = I_1 J_4 - J_5 \implies J_5 = K_3 - I_1 J_4\ .$$

Thus, we may replace J_5 by K_3 to obtain a complete polyconvex system of invariants for transversal isotropy with two preferred directions.

Lemma A.6.1 (System of Polyconvex Anisotropic Invariants for Transversal Isotropy)**.** *The set of anisotropic invariants given by*

$$\mathcal{I}^{\text{pc}}_{\text{aniso}} = \{I_1,\ I_2,\ I_3,\ J_4^{(k)},\ K_3^{(k)}\}, k = 1, 2\ ,$$

is a complete ***system of polyconvex anisotropic invariants****.*

Proof. For a more in-depth exposition, see the dissertations by Balzani [10] and Ebbing [42] and the extensive references therein. □

A.7 Incompressible Materials

Certain materials, e.g., water or biological tissue containing a high percentage of water are barely compressible. It is a common approach to model such materials as *perfectly* incompressible materials. This amounts to imposing a highly nonlinear and non-convex[6] volume constraint. We recall that a deformation mapping $\varphi : \Omega \to \Omega_{\text{def}}$ is volume preserving if and only if its deformation gradient field satisfies

$$\det[F] = \det[\nabla\varphi] = 1 \,.$$

We consider constraint functions

$$C^{\text{vol}} : \mathbb{R}^+_0 \to \mathbb{R}^+_0, \quad \text{satisfying} \quad C^{\text{vol}}(\det[F]) = 0 \iff \det[F] = 1 \,.$$

There are multiple standard approaches to incorporate constraints in a variational setting. We want to mention:

1. Penalty methods [**inexact/asymptotically exact**]
2. Lagrange multiplier methods [**exact**][**dual**]
3. Augmented Lagrange multiplier methods [**exact**][**dual + penalty**]
4. Structure preserving methods [**exact**][**primal**].

Let us give an example for a volume penalty function. Note that this function is also used for the arterial wall models on which our computations in Chapter 3 have been based.

Example A.7.1 (A Volume Constraint and Penalty Function)**.** *For the penalization of volume change we introduce the family of functions*

$$P^{\text{vol}}_\varepsilon : \mathbb{R}^+ \to \mathbb{R}^+_0, \quad P^{\text{vol}}_\varepsilon(x) := \left(x^\varepsilon + \frac{1}{x^\varepsilon} - 2 \right) . \tag{A.7.37}$$

For $\varepsilon \geq 1$ the constraint function $P^{\text{vol}}_\varepsilon$ is convex and $x = 1$ is the unique global minimizier realizing zero energy. Thus,

$$P^{\text{vol}}_\varepsilon(\det[F]) = 0 \quad \iff \quad \det[F] = 1 \,.$$

The parameter ε allows to tune the steepness of the energy valley.

In what follows we consider constraint functions $C^{\text{vol}} : \mathbb{R}^+_0 \to \mathbb{R}$ with the property

$$C^{\text{vol}}(\det[F]) = 0 \quad \iff \quad \det[F] = 1 \,.$$

We shall be satisfied to introduce these methods for the setting without configuration-dependent loads, i.e., we assume that the Euler-Lagrange equations derive from a potential.

The Penalty method: To inexactly enforce the incompressibility constraint the internal strain potential can be augmented, e.g., by a linearly weighted volume penalty term. This gives then

$$\Pi_\nu(\varphi) = \int_\Omega \Psi(F) + \nu\, P^{\text{vol}}_\varepsilon(\det[F])\ \text{dV} \quad - \Pi^{\text{ext}}_{(\text{cons})}(\varphi). \tag{A.7.38}$$

[6] Non-convex in the entries F_{ij} of the deformation gradient F.

In this approach $C^{\text{vol}} \geq 0$ is in general chosen such that it attains a global minimum for $\det[F] = 1$. The penalty weight $\nu \geq 0$ is a parameter for a corresponding family of minimization problems

$$\Pi_\nu(\varphi_{\min}(\nu)) \quad = \quad \inf_{\varphi \in \mathcal{A}} \Pi_\nu(\varphi) \, . \tag{A.7.39}$$

Assuming that the minimizers can be computed as a function of ν, one expects that in the limit

$$\lim_{\nu \to \infty} P_\varepsilon^{\text{vol}}(\det[F_{\min}(\nu)]) \quad \to \quad 0 \quad \Longleftrightarrow \quad \lim_{\nu \to \infty} \det[F_{\min}(\nu)] \quad \to \quad 1 \, . \tag{A.7.40}$$

Thus, the incompressiblity constraint is at best *asymptotically* satisfied. In most practical applications, one resorts to some large fixed value for the penalty parameter ν, i.e., one does not study the limit process. Although, especially in numerical computations, one is in general satisfied once the discretized penalty is satisfied up to a prescribed numerical tolerance, this can turn out to be impossible. In general the problem is increasingly ill-conditioned for increasing penalty weights. The penalty approach is nonetheless a popular approach in practice due to the simplicity of the implementation. Moreover, this approach is *well suited for quasi-incompressible* materials for which the volume constraint need not be exactly satisfied. In the latter case it is reasonable and possible to fit the penalty parameters from experimental data, see e.g. Brands, Klawonn, Rheinbach, and Schröder [23] and Brinkhues, Klawonn, Rheinbach, and Schröder [25]. In the present thesis, we shall focus on quasi-incompressible material behavior realized via a penalty approach. The penalty parameters have been fitted to experimental data in Brands, Klawonn, Rheinbach, and Schröder [23] by a least squares approach.

The Lagrange multiplier method: As in the finite dimensional setting, the introduction of a Lagrange multiplier λ allows to reformulate the constrained as an unconstrained optimization problem. It is then possible to satisfy the incompressibility constraint *exactly* using a Lagrange density λ for the volume constraint. Here, λ is considered as a **dual** quantity in the strain energy potential. This leads to a two-field formulation

$$\Pi^{\text{int}} : \mathcal{A}_\varphi \times \mathcal{A}_\lambda \to \mathbb{R}, \quad \Pi^{\text{int}}(\varphi, \lambda) = \int_\Omega \Psi(F) - \lambda \, C^{\text{vol}}(\det[F]) \; \text{dV}. \tag{A.7.41}$$

This unconstrained variational problem in (φ, λ) leads to two Euler-Lagrange equations. Both have to be satisfied by configurations in equilibrium which leads to a saddle point problem

$$\Pi(\varphi_{\text{crit}}, \lambda_{\text{crit}}) = \inf_{\varphi \in \mathcal{A}_\varphi} \sup_{\lambda \in \mathcal{A}_\lambda} \left\{ \Pi^{\text{int}}(\varphi, \lambda) \quad - \quad \Pi^{\text{ext}}_{(\text{cons})}(\varphi) \right\} \tag{A.7.42}$$

The Augmented Lagrange multiplier method: The so-called augmented Lagrange-method is a combination of the penalty-method with the Lagrange-multiplier method, which is obtained by adding an additional penalty term to the Lagrange functional

$$\Pi_\nu^{\text{int}} : \mathcal{A}_\varphi \times \mathcal{A}_\lambda \to \mathbb{R} : \quad \Pi_\nu^{\text{int}}(\varphi, \lambda) = \int_\Omega \Psi(F) - \lambda \, C^{\text{vol}}(\det[F]) + \nu \, P_\varepsilon^{\text{vol}}(\det[F]) \; \text{dV}. \tag{A.7.43}$$

As in the penalty approach there is a whole family of variational problems and we determine critical points of the saddle point problem

$$\Pi_\nu(\varphi_{\text{crit}}(\nu), \lambda_{\text{crit}}(\nu)) = \inf_{\varphi \in \mathcal{A}_\varphi} \sup_{\lambda \in \mathcal{A}_\lambda} \left\{ \Pi_\nu^{\text{int}}(\varphi, \lambda) \quad - \quad \Pi^{\text{ext}}_{(\text{cons})}(\varphi) \right\} \, . \tag{A.7.44}$$

Again, the augmented penalty term vanishes *asymptotically* once the dual constraint is met. However, it does still contribute and can be used to update the Lagrange multiplier λ. The augmented

approach was shown to be very effective for the soft biological tissue material models for arteria walls which we consider in the present text; see Brinkhues, Klawonn, Rheinbach, and Schröder [25] and also the dissertation by Brinkhues [24]. More precisely, the augmented approach was compared with the penalty and the standard Lagrange-multiplier approaches and it has been found to be a superior approach. This is not implemented in the presented solver `mparfeap`. The penalty parameters have been fitted to experimental data in Brinkhues, Klawonn, Rheinbach, and Schröder [25].

Structure Preserving Methods: In a purely primal approach the volume constraint is exactly satisfied by an appropriate parametrization of the constraint manifold. In our context, this amounts to consider the configuration space for a perfectly incompressible elastic body given by

$$\mathcal{C}_\Omega^{\mathrm{vol}} := \{\varphi \in \mathcal{C}_\Omega^{\mathrm{elas}} : \det[F] = 1\} .$$

It is possible to model this exactly on the discrete level using particular structure preserving discretization methods. See, e.g., the recent approach to incompressible linear elasticity by Angoshtari and Yavari [4] and references therein.

A.8 First and Second Variation of the Internal Strain Energy

In this section, we derive the symmetry of the bilinear form associated to the second variation of the total interior strain energy. This was exploited in `mparfeap` to conserve memory and it may be of some interest, although this is standard. We have also found that it is helpful to be aware of this symmetry in order to avoid implementation mistakes.

A second goal is to display the differences in the implementation arising from basing the formulation of the second variation on the first or second Piola-Kirchhoff stress tensor, respectively. There is in fact no difference from the point of physics that are modeled, but one obtains different expressions. We shall restrict ourselves to a formal derivation of the first and second variation of the total interior strain energy, but do this in a more detailed way than it is usually done in the literature. In turn, let us suppose that all Fréchet- and Gâteaux-derivatives which appear in the following do exist and are continuous. For simplicity of exposition, all issues regarding regularity of the mappings involved shall simply be neglected. Further, we write the equilibrium equations with respect to the reference configuration Ω and consider variations only on linear spaces of functions.[7]

We recall that in the hyperelastic framework, the internal strain energy is given by an integral

$$\Pi^{\mathrm{int}}(\varphi) := \int_\Omega \Psi(F)\,\mathrm{dV} := \int_\Omega \Psi(\nabla\varphi)\,\mathrm{dV} .$$

Due to objectivity, there is a $\widehat{\Psi}(C)$ satisfying

$$\forall F \in \mathrm{GL}^+(3,\mathbb{R}): \quad \Psi(F) = \widehat{\Psi}(C(F)) = \widehat{\Psi}(F^T F).$$

[7] In particular, it is then sufficient to consider directional derivatives of the potential along linear smooth curves γ_{φ_0} in the space of admissible configurations, i.e.,

$$\gamma_{\varphi_0} : (-\varepsilon,\varepsilon) \to \mathcal{A}, \quad \gamma_{\varphi_0}(t) := \varphi_0 + t\,\delta\varphi , \quad \delta\varphi \in T_{\varphi_0}\mathcal{A}. \tag{A.8.45}$$

The matrix representations of the first and second Piola-Kirchhoff stress tensors $S_1(F)$ and $S_2(C)$ are determined as follows:

$$\forall X \in \mathrm{M}(3,\mathbb{R}): \quad \big\langle S_1(F_0),\, X\big\rangle = d_F\Psi|_{F_0}(X) = \sum_{i,j=1}^{3} \left.\frac{\partial \Psi(F)}{\partial F_{ij}}\right|_{F_0} X_{ij}\,, \tag{A.8.46}$$

$$\forall X \in \mathrm{M}(3,\mathbb{R}): \quad \big\langle S_2(C_0),\, X\big\rangle = 2\, d_C\, \widehat{\Psi}\,|_{C_0}(X) = 2\sum_{i,j=1}^{3} \left.\frac{\partial\, \widehat{\Psi}(C)}{\partial C_{ij}}\right|_{C_0} X_{ij}\,. \tag{A.8.47}$$

The second relation is called the Doyle-Erickson formula with respect to the reference configuration. Next, we compute the first and second variations of the strain energy functional. Due to the Doyle-Erickson formula, there are two natural equivalent formulations in terms of the first and second Piola-Kirchhoff stresses. These are associated to variations of the internal strain energy based on the two equivalent representations of the local strain energy density

$$\Pi^{\mathrm{int}}(\varphi) := \int_\Omega \Psi(F)\ \mathrm{dV} = \int_\Omega \widehat{\Psi}(C(F))\ \mathrm{dV}\,.$$

Both model the same physics, but they lead to different expressions.

Lemma A.8.1 (First Variation of the Internal Strain Energy)**.** *The first variation of the internal strain energy $\Pi^{\mathrm{int}}(\varphi)$ formulated in terms of $\Psi(F)$ is given by*

$$d_\varphi\Pi^{\mathrm{int}}|_{\varphi_0}(\chi) = \int_\Omega d_F\Psi(F)|_{F_0}(\nabla\chi) = \int_\Omega \big\langle S_1(F_0),\, \nabla\chi\big\rangle\ dV\,.$$

The equivalent formulation in terms of $\widehat{\Psi}(C(F))$ is given by

$$d_\varphi\Pi^{\mathrm{int}}|_{\varphi_0}(\chi) = \int_\Omega d_C\,\widehat{\Psi}(C)|_{C_0} \circ d_F C(F)|_{F_0}(\nabla\chi) \qquad \left[= \int_\Omega \big\langle \underbrace{F S_2(C_0))}_{=S_1(F_0)},\, \nabla\chi\big\rangle\ dV\right].$$

Proof. We compute the first variation for the representation associated to $\Psi(F)$ to obtain

$$\begin{aligned}
d_\varphi\Pi^{\mathrm{int}}|_{\varphi_0}(\chi) &:= \left.\frac{d}{dt}\right|_{t=0} \Pi^{\mathrm{int}}(\varphi_0 + t\chi) && \text{(A.8.48)}\\
&= \int_\Omega \left.\frac{d}{dt}\right|_{t=0} \Psi(\nabla(\varphi_0 + t\chi))\ \mathrm{dV} && \text{(A.8.49)}\\
&= \int_\Omega d_F\Psi(F)|_{\nabla(\varphi_0+t\chi)|_{t=0}} \left(\left.\frac{d}{dt}\right|_{t=0} \nabla\,(\varphi_0 + t\chi)\right)\ \mathrm{dV} && \text{(A.8.50)}\\
&= \int_\Omega d_F\Psi(F)|_{\nabla(\varphi_0)} \left(\nabla \left.\frac{d}{dt}\right|_{t=0} \varphi_0 + t\chi\right)\ \mathrm{dV} && \text{(A.8.51)}\\
&= \int_\Omega d_F\Psi(F)|_{F_0}(\nabla\chi)\ \mathrm{dV} && \text{(A.8.52)}\\
&= \int_\Omega \big\langle S_1(F_0),\, \nabla\chi\big\rangle\ \mathrm{dV}\,. && \text{(A.8.53)}
\end{aligned}$$

This shows the first part of the claim. Note that the calculation for the second part can be carried out in complete analogy, with the application of the chain rule carried out as in the proof of Lemma A.3.12. □

Lemma A.8.2 (Second Variation of the Internal Strain Energy)**.** *The second variation of the internal strain energy* $\Pi^{\text{int}}(\varphi)$ *formulated in terms of* $\Psi(F)$ *is given by*

$$d^2_\varphi \Pi^{\text{int}}|_{\varphi_0}(\chi,\eta) = \int_\Omega d^2_F \Psi|_{F_0}(\nabla\chi, \nabla\eta)\ dV\,. \tag{A.8.54}$$

This bilinear form is symmetric for all $\varphi_0 \in \mathcal{A}$.

Proof. First, we show symmetry of the second derivative of the local strain energy density $\Psi(F)$ taken with respect to F. Starting from the second derivative of the strain energy density, we note

$$d^2_F \Psi|_{F_0}(X,Y) = \sum_{i,j,k,l=1}^{3} \frac{\partial^2 \Psi(F)}{\partial F_{ij}\partial F_{kl}} X_{ij} Y_{kl}\ .$$

Changing the differentiation order and relabeling, we get

$$\begin{aligned} d^2_F \Psi|_{F_0}(X,Y) &= \sum_{i,j,k,l=1}^{3} \frac{\partial^2 \Psi(F)}{\partial F_{ij}\partial F_{kl}} X_{ij} Y_{kl} = \sum_{i,j,k,l=1}^{3} \frac{\partial^2 \Psi(F)}{\partial F_{kl}\partial F_{ij}} Y_{kl} X_{ij} = \sum_{i,j,k,l=1}^{3} \frac{\partial^2 \Psi(F)}{\partial F_{ij}\partial F_{kl}} Y_{ij} X_{kl} \\ &= d^2_F \Psi|_{F_0}(Y,X)\ . \end{aligned} \tag{A.8.55}$$

Hence, the second derivative of the local strain energy density is symmetric with respect to X and Y. The bilinear form associated to the second variation of the strain energy potential is thus symmetric, since

$$d^2_\varphi \Pi^{\text{int}}|_{\varphi_0}(\chi,\eta) = \int_\Omega d^2_F \Psi|_{\nabla\varphi_0}(\nabla\chi,\nabla\eta)\ \mathrm{dV} = \int_\Omega d^2_F \Psi|_{\nabla\varphi_0}(\nabla\eta,\nabla\chi)\ \mathrm{dV} = d^2_\varphi \Pi^{\text{int}}|_{\varphi_0}(\eta,\chi) \tag{A.8.56}$$

for all admissible $\varphi_0 \in \mathcal{A}$. □

Note that this immediately implies that the associated finite element stiffness matrix $K^{\text{int}}(\varphi)$ will also be symmetric when using a standard Galerkin-ansatz. It may be confusing on first sight that the second variation of the strain energy can be equivalently expressed in terms of $\Psi(F)$ and $\widehat{\Psi}(C(F))$, i.e., in terms of S_1 or $FS_2 = S_1$. To render the implications clear, it is instructive to derive the following identity.

Lemma A.8.3. *We have*

$$d^2_F\, \widehat{\Psi}(C(F))|_{F_0}(X,Y) = \left\langle S_2(C_0),\, \mathrm{sym}\left[X^T Y\right] \right\rangle + 4\, d^2_C\, \widehat{\Psi}(C)|_{C_0}(\mathrm{sym}\left[F_0^T X\right], \mathrm{sym}\left[F_0^T Y\right])\ ,$$

with $F_0 := \nabla\varphi_0$ *and* $C_0 := C(F_0) = \nabla\varphi_0^T\, \nabla\varphi_0$. *Both terms on the right hand side are symmetric with respect to* X *and* Y.

Proof. Applying the chain rule multiple times, we obtain

$$
\begin{aligned}
&\frac{d}{dt}\Big|_{t=0} \big\langle (F_0+tY) S_2(C(F_0+tY)),\, X \big\rangle \\
&= \big\langle \frac{d}{dt}\Big|_{t=0} (F_0+tY) S_2(C(F_0+tY)),\, X \big\rangle + \big\langle\, (F_0+tY) S_2(C(F_0+tY))|_{t=0}\,,\, \frac{d}{dt}\Big|_{t=0} X \big\rangle \\
&= \big\langle \left[\frac{d}{dt}\Big|_{t=0} (F_0+tY) \right] S_2(C(F_0+tY))|_{t=0} + (F_0+tY)|_{t=0} \left[\frac{d}{dt}\Big|_{t=0} S_2(C(F_0+tY)) \right],\, X \big\rangle \\
&= \big\langle Y S_2(C_0) + F_0 \, \frac{d}{dt}\Big|_{t=0} S_2(C(F_0+tY)),\, X \big\rangle \\
&= \big\langle Y S_2(C_0)),\, X \big\rangle + \big\langle \, \frac{d}{dt}\Big|_{t=0} S_2(C(F_0+tY)),\, F_0^T X \big\rangle \\
&= \big\langle S_2(C_0),\, Y^T X \big\rangle + \big\langle \, \frac{d}{dt}\Big|_{t=0} S_2(C(F_0+tY)),\, \mathrm{sym}\left[F_0^T X\right] \big\rangle \\
&= \big\langle S_2(C_0),\, \mathrm{sym}\left[X^T Y\right] \big\rangle + \frac{d}{dt}\Big|_{t=0} d_C\, \widehat{\Psi}(C)|_{C(F_0+tY)} \left(2\,\mathrm{sym}\left[F_0^T X\right]\right) \\
&= \big\langle S_2(C_0),\, \mathrm{sym}\left[X^T Y\right] \big\rangle + d_C^2\, \widehat{\Psi}(C)\Big|_{C(F_0)} \left(2\,\mathrm{sym}\left[F_0^T X\right],\, \frac{d}{dt}\Big|_{t=0} C(F_0+tY)\right) \\
&= \big\langle S_2(C_0),\, \mathrm{sym}\left[X^T Y\right] \big\rangle + 4\; d_C^2\, \widehat{\Psi}(C)\Big|_{C_0} \left(\mathrm{sym}\left[F_0^T X\right], \mathrm{sym}\left[F_0^T Y\right]\right) \,.
\end{aligned}
$$

□

We summarize that there are two equivalent possibilities to write (and implement) the second variation of the strain energy potential.

Equivalent Representations of the Second Variation Internal Strain Energy

$$
d_\varphi^2 \Pi^{\mathrm{int}}|_{\varphi_0}(\chi,\eta) = \int_\Omega d_F^2 \Psi(F)|_{F_0}(\nabla\chi, \nabla\eta)\,\mathrm{dV} \tag{A.8.57}
$$

and

$$
\begin{aligned}
d_\varphi^2 \Pi^{\mathrm{int}}|_{\varphi_0}(\chi,\eta) &= \int_\Omega d_F^2\, \widehat{\Psi}(C(F))|_{F_0}(\nabla\chi, \nabla\eta)\,\mathrm{dV} && \text{(A.8.58)} \\
&= \int_\Omega \underbrace{\big\langle S_2(C(F_0)),\, \mathrm{sym}\left[\nabla\chi^T \nabla\eta\right] \big\rangle}_{\text{geometric part}}\; \mathrm{dV} && \text{(A.8.59)} \\
&\quad + 4 \int_\Omega \underbrace{d_C^2\, \widehat{\Psi}(C)|_{C(F_0)} \left(\mathrm{sym}\left[F_0^T \nabla\chi\right], \mathrm{sym}\left[F_0^T \nabla\eta\right]\right)}_{\text{material part}}\; \mathrm{dV}\,. && \text{(A.8.60)}
\end{aligned}
$$

Each of the two integrands is symmetric with respect to the variations $\chi, \eta T_{\varphi_0}\mathcal{A}$. This observation is independent of the specific deformation mapping φ_0 where the linearization was carried out.

The material models implemented in the stock distribution of FEAP are implemened using the second formulation based on $\widehat{\Psi}(C)$, i.e., on the second Piola-Kirchhoff tensor $S_2(C)$. This can be inferred from Taylor and the FEAP team [128], where also the naming conventions for the material and geometric part of the second variation is used. In practice this allows to base the implementation on the first and second derivatives of the invariants with respect to C.

Let us shortly discuss some well-known properties of the second variation of the internal strain energy. These are in part inherited by the finite element systems which are solved in a Newton-Krylov solver for finite elasticity.

Remark A.8.4 (Some Properties of the Second Variation of Π^{int})**.** *Globally positive definite second derivatives are a property of convex potentials. The linearization of the equilibrium equations, i.e., the second variation of the strain energy, is not coercive in general. Global coercivity implies unphysical behavior, because global convexity of the strain energy functional precludes the possibility of bifurcations including buckling. This is mentioned, e.g., in the context of polyconvex strain energy densities in the dissertation of Balzani [10]. We summarize that the finite element stiffness matrices obtained from the second variation of the internal strain energy potential using a standard Galerkin-ansatz for the displacement are symmetric, but possibly indefinite.*

Finally, let us study the relation to linear elasticity. In the reference configuration, i.e., for $\varphi_0 = \mathrm{id}_\Omega$, we have $C_0 = F_0 = \mathbb{1}$. For this special case one obtains

$$d_F^2 \Psi(F)\big|_{\mathbb{1}}\,(X,Y) = \big\langle \mathbb{c}X,\, Y \big\rangle = \left. d_C^2\, \widehat{\Psi}(C) \right|_{\mathbb{1}}\,(X,Y)\,. \tag{A.8.61}$$

Since the body is stress-free in the reference configuration

$$\big\langle \underbrace{S_2(\mathbb{1})}_{=0},\, \mathrm{sym}\left[X^T Y\right] \big\rangle = 0\,. \tag{A.8.62}$$

Here, $\mathbb{c}$ denotes the fourth order elasticity tensor.

A.9 The $\overline{\mathrm{F}}$-approach to Incompressibility

A naive displacement based approach for incompressible elasticity often leads to numerical problems due to the so-called "volume locking"-effect. It is often observed that the volume constraint inhibits the convergence of the numerical finite element solution.

Different methods have been proposed in the literature to improve the numerical approximation of incompressible material formulations. For the presented computations in Chapter 3, we use an $\overline{\mathrm{F}}$-method due to Simo [122, Sec. 45].[8] We note that some $\overline{\mathrm{F}}$-methods are susceptible to the "hourglassing"-effect. This is not the case for the approach due to Nagtegaal taken here. For a survey and comparisons of different $\overline{\mathrm{F}}$-methods, we refer the interested reader to the thesis of Freischläger [54]. For the approach taken here, we follow the expositions of Brinkhues [24] and Brands [20]. It shall suffice here to outline the $\overline{\mathrm{F}}$-approach used in the simulations presented in Brands, Klawonn, Rheinbach, and Schröder [23] and Brinkhues, Klawonn, Rheinbach, and Schröder [25]. Note that the $\overline{\mathrm{F}}$-approach was used for investigations regarding the scalability of FETI-DP in biomechanics applications by Klawonn and Rheinbach [73].

Consider a multiplicative decomposition of the deformation gradient $F := \nabla\varphi$ into a volumetric and an isochoric part, given by

$$F = F^{\mathrm{vol}}(F) \cdot F^{\mathrm{isc}}(F)\,.$$

In this context it is convenient to introduce $J := \det[F]$. We introduce the mappings

$$F^{\mathrm{vol}} : \mathrm{GL}^+(3,\mathbb{R}) \to \mathbb{R}_0^+ \cdot \mathbb{1}\,, \qquad F^{\mathrm{vol}}(F) := J^{1/3}\,\mathbb{1}\,, \tag{A.9.63}$$

$$F^{\mathrm{isc}} : \mathrm{GL}^+(3,\mathbb{R}) \to \mathrm{SL}(3,\mathbb{R})\,, \qquad F^{\mathrm{isc}}(F) := J^{-1/3}\,F\,. \tag{A.9.64}$$

Note that both maps are surjective, multiplicative and idempotent functions.

[8] The $\overline{\mathrm{F}}$-methods are also referred to as $\overline{\mathrm{B}}$-methods.

Definition A.9.1 (F-bar)**.** *Let $F \in \mathrm{GL}^+(3,\mathbb{R})$ be a deformation gradient and let $\Theta > 0$ be a volumetric dilatation factor. Then the* ***modified deformation gradient*** *is defined as*

$$\overline{\mathrm{F}}(F,\Theta) := \Theta^{1/3}\, F^{\mathrm{isc}}(F)\,. \tag{A.9.65}$$

The following identity is then immediate

$$\det\left[\,\overline{\mathrm{F}}\,\right] = \det\left[\left(\frac{\Theta}{J}\right)^{1/3} F\right] = \left(\frac{\Theta}{J}\right) J = \Theta\,. \tag{A.9.66}$$

Definition A.9.2 (F-bar Three-field Hu-Washizu Potential)**.** *The three-field Hu-Washizu functional for the $\overline{\mathrm{F}}$-bar method is given by*

$$\Pi^{\mathrm{int}}(\varphi,\Theta,\lambda) = \int_\Omega \Psi(\overline{\mathrm{F}}(F,\Theta)) + \lambda\,(J-\Theta)\;dV \tag{A.9.67}$$

For a P_0-finite element ansatz in Θ and λ, this is equivalent to the $\overline{\mathrm{F}}$-approach due to Nagtegaal. To see this, one departs from the first variation with respect to λ to obtain

$$d_\lambda \Pi^{\mathrm{int}}\big|_{(\varphi_0,\Theta_0,\lambda_0)}\,(\delta\lambda) = \int_\Omega \delta\lambda\,(J_0-\Theta_0)\,\mathrm{dV}\,. \tag{A.9.68}$$

Since Θ_0 is constant on every element $T \in \mathcal{T}$, we have

$$\Theta_0 = \frac{\int_T J_0\; dV}{\int_T 1\; dV} = \frac{\mathrm{Vol}(\varphi_0(T))}{\mathrm{Vol}(T)}\,. \tag{A.9.69}$$

Thus, for any solution of the Euler-Lagrange equation, Θ_0 is the average volume distortion per element in the configuration φ_0. This is exactly the choice for the ansatz for the volume dilatation in the approach due to Nagtegaal. In this setting, it is possible to reduce the second variation via static condensation. In particular, there are no additional degrees of freedom as compared to the displacement formulation. Although it is not obvious, we repeat that this still leads to a *symmetric* second variation of the internal strain energy. Note that the symmetry may be lost for some $\overline{\mathrm{F}}$-methods, e.g., the Souza-Neto approach, see again Freischläger [54].

Contributions to Extended Continuum Theories

Although not the subject of the current thesis, we want to shortly mention some contributions of P. Neff and the present author to the mechanics of generalized continua which were realized prior and in parallel to this dissertation. Some work on theories with rotational degress of freedom, in particular Cosserat theory, was published in Fischle, Neff, and Münch [51], Neff, Fischle, and Münch [102] and Neff, Jeong, and Fischle [103]. In his diploma thesis, the present author computed energy minimizing Cosserat rotations using certain symmetry reductions. Another contribution is a generalization thereof to logarithmic strain measures for unitary matrices in Neff, Nakatsukasa, and Fischle [104].

List of Figures

List of Tables

References

[1] Mark F. Adams, Harun H. Bayraktar, Tony M. Keaveny, and Panayiotis Papadopoulos. Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 34. IEEE Computer Society, 2004. 46, 131, 137

[2] Howard G. Allen and Philip S. Bulson. Background to buckling, 1980. 16

[3] Eugene L. Allgower and Kurt Georg. *Numerical Continuation Methods*, volume 13. Springer-Verlag Berlin, 1990. 31

[4] Arzhang Angoshtari and Arash Yavari. A geometric structure-preserving discretization scheme for incompressible linearized elasticity. *Computer Methods in Applied Mechanics and Engineering*, 2013. 166

[5] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997. ii, 37

[6] Satish Balay, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.4, Argonne National Laboratory, 2013. URL `http://www.mcs.anl.gov/petsc`. 37

[7] Satish Balay, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, and Hong Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2014. URL `http://www.mcs.anl.gov/petsc`. ii, 37

[8] John M. Ball. Convexity conditions and existence theorems in nonlinear elasticity. *Archive for rational mechanics and Analysis*, 63(4):337–403, 1976. 3, 162

[9] John M. Ball. Constitutive Inequalities and Existence Theorems in Nonlinear Elastostatics. In *Nonlinear Analysis and Mechanics: Heriot-Watt Symposium*, volume 1, pages 187–241, 1977. 3

[10] Daniel Balzani. *Polyconvex Anisotropic Energies and Modeling of Damage Applied to Arterial Walls*. PhD thesis, Universität Duisburg-Essen, 2006. Bericht Nr. 2, Institut für Mechanik, Fakultät für Ingenieurwissenschaften. 2, 3, 5, 12, 13, 16, 161, 162, 163, 170

[11] Daniel Balzani, Patrizio Neff, Jörg Schröder, and Gerhard A. Holzapfel. A Polyconvex Framework for Soft Biological Tissues. Adjustment to Experimental Data. *International Journal of Solids and Structures*, 43(20):6052–6070. 2, 10, 89, 90

[12] Daniel Balzani, Dominik Brands, Axel Klawonn, and Oliver Rheinbach. Large-scale simulation of arterial walls: mechanical modeling. *PAMM*, 7(1):4020017–4020018, 2007. ISSN 1617-7061. doi: 10.1002/pamm.200700400. URL http://dx.doi.org/10.1002/pamm.200700400. i, 88

[13] Daniel Balzani, Dominik Brands, Axel Klawonn, Oliver Rheinbach, and Jörg Schröder. On the mechanical modeling of anisotropic biological soft tissue and iterative parallel solution strategies. *Archive of Applied Mechanics*, 80(5):479–488, 2010. i, 88

[14] Daniel Balzani, Dirk Böse, Dominik Brands, Raimund Erbel, Axel Klawonn, Oliver Rheinbach, and Jörg Schröder. Parallel simulation of patient-specific atherosclerotic arteries for the enhancement of intravascular ultrasound diagnostics. *Engineering Computations*, 29(8): 888–906, 2012. 2, 3, 7, 93

[15] Wolfgang Bangerth and Timo Heister. Quo Vadis, Scientific Software? *SIAM News*, January 2014. i

[16] Zdeněk P. Bažant and Luigi Cedolin. *Stability of Structures: Elastic, Inelastic, Fracture and Damage Theories*. World Scientific, 2010. 16

[17] Manoj Bhardwaj, David Day, Charbel Farhat, Michel Lesoinne, Kendall H. Pierson, and Daniel Rixen. Application of the FETI method to ASCI problems - scalability results on one thousand processors and discussion of highly heterogeneous problems. *Internat. J. Numer. Methods Engrg.*, 47:513–535, 2000. 19

[18] David Bindel. MATFEAP, 2011. http://www.cs.cornell.edu/~bindel/sw/matfeap. 46, 71

[19] Jean-Paul Boehler. Introduction to the invariant formulation of anisotropic constitutive equations. In J.P. Boehler, editor, *Applications of Tensor Functions in Solid Mechanics*, number 292 in CISM Course. Springer, San Francisco, CA, 1987. 160

[20] Dominik Brands. *Geometrical Modeling and Numerical Simulation of Heterogeneous Materials*. PhD thesis, Fakultät für Ingenieurwissenschaften, Abteilung Bauwissenschaften, Universität Duisburg-Essen, Essen, Germany, 2012. i, 2, 3, 5, 88, 99, 100, 101, 170

[21] Dominik Brands, Axel Klawonn, Oliver Rheinbach, and Jörg Schröder. On the visualization of patient specific arterial geometries using the virtual histology. *PAMM*, 8(1):10169–10170, 2008. ISSN 1617-7061. doi: 10.1002/pamm.200810169. URL http://dx.doi.org/10.1002/pamm.200810169. i, 88

[22] Dominik Brands, Jörg Schröder, Axel Klawonn, Oliver Rheinbach, Dirk Böse, and Raimund Erbel. Numerical Simulations of Arterial Walls Based on Ivus-data. *PAMM*, 9(1):75–78, 2009. ISSN 1617-7061. doi: 10.1002/pamm.200910020. URL http://dx.doi.org/10.1002/pamm.200910020. 2, 7

[23] Dominik Brands, Axel Klawonn, Oliver Rheinbach, and Jörg Schröder. Modelling and convergence in arterial wall simulations using a parallel FETI solution strategy. *Comput. Meth. Biomech. Biomed. Eng.*, Vol. 11(No. 5):pp. 569–583, October 2008. i, ii, iii, iv, 2, 3, 5, 8, 9, 10, 11, 12, 13, 16, 27, 28, 80, 88, 89, 90, 91, 93, 99, 100, 101, 107, 108, 143, 162, 165, 170

[24] Sarah Brinkhues. *Modelling and simulation of arterial walls with focus on damage and residual stresses*. PhD thesis, Fakultät für Ingenieurwissenschaften, Abteilung Bauwissenschaften, Universität Duisburg-Essen, Essen, Germany, 2013. i, 3, 5, 9, 12, 13, 28, 30, 88, 89, 90, 144, 166, 170

[25] Sarah Brinkhues, Axel Klawonn, Oliver Rheinbach, and Jörg Schröder. Augmented Lagrange Methods in the Simulation of Soft Biological Tissue. *International Journal for Numerical Methods in Biomedical Engineering*, 2012. doi: http://dx.doi.org/10.1002/cnm.2504. i, ii, 2, 3, 12, 13, 28, 30, 80, 88, 89, 90, 144, 165, 166, 170

[26] Hans Bufler. Pessure Loaded Structures Under Large Deformations. *Zeitschrift für Angewandte Mathematik und Mechanik (ZAMM)*, 64(7):287–295, 1984. 155

[27] Chen-Yi Chen. PyFEAP – A Python Interface to Finite Element Analysis Program (FEAP). `www.cs.cornell.edu/~bindel/projects/pyfeap-report.pdf`. 46

[28] Philippe G. Ciarlet. *The Finite Element Method for Elliptic Problems*. North-Holland, Amsterdam, 1978. 13

[29] Philippe G. Ciarlet. *Mathematical Elasticity Volume I: Three–Dimensional Elasticity*. North-Holland, 1988. 13, 148, 150

[30] Jean-Michel Cros. A preconditioner for the Schur complement domain decomposition method. In O. Widlund I. Herrera, D. Keyes and R. Yates, editors, *Domain Decomposition Methods in Science and Engineering*, pages 373–380. National Autonomous University of Mexico (UNAM), Mexico City, Mexico, ISBN 970-32-0859-2, 2003. Proceedings of the 14th International Conference on Domain Decomposition Methods in Science and Engineering; http://www.ddm.org/DD14. 19

[31] Bernard Dacorogna. *Direct methods in the calculus of variations*, volume 78. Springer, 2008. 162

[32] Ron S. Dembo, Stanley C. Eisenstat, and Trond Steihaug. Inexact Newton Methods. *SIAM J. Numer. Anal.*, 19(2):400–408, 1982. 27

[33] John E. Dennis Jr. and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1996. 33

[34] Peter Deuflhard. *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms*. Springer Series in Computational Mathematics. 2004. ISBN 9783540210993. 17, 33, 132

[35] Clark R. Dohrmann. A preconditioner for substructuring based on constrained energy minimization. *SIAM J. Sci. Comput.*, 25(1):246–258, 2003. 19

[36] Clark R. Dohrmann. An approximate BDDC preconditioner. *Numer. Linear Algebra Appl.*, 14(2):149–168, 2007. ISSN 1070-5325. 19

[37] Clark R. Dohrmann and Richard B. LeHoucq. A primal-based penalty preconditioner for elliptic saddle point systems. *SIAM J. Numer. Anal.*, 44(1):270–282, 2006. 133

[38] Clark R. Dohrmann and Olof B. Widlund. An Overlapping Schwarz Algorithm for Almost Incompressible Elasticity. *SIAM J. Numer. Anal.*, 47(4):2897–2923, 2009. 19

[39] Clark R. Dohrmann, Axel Klawonn, and Olof B. Widlund. Domain decomposition for less regular subdomains: overlapping Schwarz in two dimensions. *SIAM J. Numer. Anal.*, 46(4): 2153–2168, 2008. ISSN 0036-1429. 19, 24

[40] Clark R. Dohrmann, Axel Klawonn, and Olof B. Widlund. Extending theory for domain decomposition algorithms to irregular subdomains. In *Domain decomposition methods in science and engineering XVII*, volume 60 of *Lect. Notes Comput. Sci. Eng.*, pages 255–261. Springer, Berlin, 2008. doi: 10.1007/978-3-540-75199-1_29. URL http://dx.doi.org/10.1007/978-3-540-75199-1_29. 19

[41] Clark R. Dohrmann, Axel Klawonn, and Olof B. Widlund. A family of energy minimizing coarse spaces for overlapping Schwarz preconditioners. In *Domain decomposition methods in science and engineering XVII*, volume 60 of *Lect. Notes Comput. Sci. Eng.*, pages 247–254. Springer, Berlin, 2008. doi: 10.1007/978-3-540-75199-1_28. URL http://dx.doi.org/10.1007/978-3-540-75199-1_28. 19

[42] Vera Ebbing. *Design of Polyconvex Energy Functions for All Anisotropy Classes*. PhD thesis, Fakultät für Ingenieurwissenschaften, Abteilung Bauwissenschaften, Universität Duisburg-Essen, Essen, Germany, 2010. 159, 161, 163

[43] Charbel Farhat. A Lagrange multiplier based on divide and conquer finite element algorithm. *J. Comput. System Engrg.*, 2:149–156, 1991. 22

[44] Charbel Farhat and Francois-Xavier Roux. A method of Finite Element Tearing and Interconnecting and its parallel solution algorithm. *Internat. J. Numer. Methods Engrg.*, 32: 1205–1227, 1991. 19, 22

[45] Charbel Farhat and Francois-Xavier Roux. An Unconventional Domain Decomposition Method for an Efficient Parallel Solution of Large-Scale Finite Element Systems. *SIAM J. Sc. Stat. Comput.*, 13:379–396, 1992. 22

[46] Charbel Farhat and Francois-Xavier Roux. Implicit parallel processing in structural mechanics. In J. Tinsley Oden, editor, *Computational Mechanics Advances*, volume 2(1), pages 1–124. North-Holland, 1994. 19

[47] Charbel Farhat, Po-Shu Chen, Franck Risler, and Francois-Xavier Roux. A unified framework for accelerating the convergence of iterative substructuring methods with Lagrange multipliers. *Internat. J. Numer. Methods Engrg.*, 42:257–288, 1998. 19

[48] Charbel Farhat, Michel Lesoinne, and Kendall H. Pierson. A scalable dual-primal domain decomposition method. *Numer. Linear Algebra Appl.*, 7:687–714, 2000. 19, 22, 24

[49] Charbel Farhat, Michel Lesoinne, Patrick Le Tallec, Kendall H. Pierson, and Daniel J. Rixen. FETI-DP: A dual-primal unified FETI method – part I: A faster alternative to the two-level FETI method. *Internat. J. Numer. Methods Engrg.*, 50:1523–1544, 2001. 19, 22, 26

[50] Charbel Farhat, Jing Li, and Philip Avery. A FETI-DP method for the parallel iterative solution of indefinite and complex-valued solid and shell vibration problems. *International Journal for Numerical Methods in Engineering*, 63(3):398–427, 2005. ISSN 1097-0207. doi: 10.1002/nme.1282. URL http://dx.doi.org/10.1002/nme.1282. 27

[51] Andreas Fischle, Patrizio Neff, and Ingo Münch. Symmetric Cauchy stresses do not imply symmetric Biot strains in weak formulations of isotropic hyperelasticity with rotational degrees of freedom. *PAMM*, 8(1):10419–10420, 2008. 159, 171

[52] Andreas Fischle, Axel Klawonn, Oliver Rheinbach, and Jörg Schröder. Parallel simulation of biological soft tissue. *PAMM*, 12(1):767–768, 2012. i, 88

[53] Yannis Fragakis and Manolis Papadrakakis. The mosaic of high performance domain decomposition methods for structural mechanics: Formulation, interrelation and numerical efficiency of primal and dual methods. *Comput. Methods Appl. Mech. Engrg*, 192(35–36): 3799–3830, 2003. 19

[54] Christian Freischläger. *Konzepte zur Formulierung versteifungsfreier Volumenelemente*. PhD thesis, Universität Fridericiana Karlsruhe, 2000. 16, 170, 171

[55] Sabrina Gippert. *Domain Decomposition Methods for Elastic Materials with Compressible and Almost Incompressible Components*. PhD thesis, Universität Duisburg-Essen, Fakultät für Mathematik, Essen, Germany, 2012. i, 28, 88, 89, 133

[56] Sabrina Gippert, Axel Klawonn, and Oliver Rheinbach. Analysis of FETI-DP and BDDC for linear elasticity in 3D with almost incompressible components and varying coefficients inside subdomains. *SIAM Journal on Numerical Analysis*, 50(5):2208–2236, 2012. 28

[57] Sabrina Gippert, Axel Klawonn, and Oliver Rheinbach. *A Deflation Based Coarse Space in Dual-Primal FETI Methods for Almost Incompressible Elasticity*. Submitted to the Proceedings of the ENUMATH Conference. Lausanne, Switzerland, August 26-30, 2013. 28, 133

[58] Pierre Gosselet and Christian Rey. On a selective reuse of Krylov subspaces in Newton-Krylov approaches for nonlinear elasticity. In *Proceedings of the fourteenth international conference on domain decomposition methods*, pages 419–426, 2003. 28, 137, 144

[59] Pierre Gosselet and Christian Rey. Non-overlapping domain decomposition methods in structural mechanics. *Arch. Comput. Methods Engrg.*, 13(4):515–572, 2006. ISSN 1134-3060. 28, 144

[60] Luigi Grippo, Francesco Lampariello, and Stephano Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986. 33

[61] Bjørn Haugen. *Buckling and Stability Problems for Thin Shell Structures using High Performance Finite Elements*. PhD thesis, University of Colorado, 1994. 15

[62] Alexander Heinlein. Integration of `libfw` into `LifeV`. Personal Communication, 2013. iv, 143

[63] Gerhard A. Holzapfel. *Nonlinear Solid Mechanics: A Continuum Approach for Engineering*. John Wiley & Sons, 2000. ISBN 9780471823193. 148

[64] Gerhard A. Holzapfel, Christian T. Gasser, and Raymound W. Ogden. A new constitutive framework for arterial wall mechanics and a comparative study of material models. *Journal of Elasticity*, 61:1–48, 2000. 5, 10, 90

[65] Gerhard A. Holzapfel, Christian T. Gasser, and Raymound W. Ogden. Comparison of a multi-layer structural model for arterial walls with a fung-type model, and issues of material stability. *Journal of Biomechanical Engineering*, 126:264–275, 2004. 10

[66] Eva Kanso, Marino Arroyo, Yiying Tong, Arash Yavari, Jerrold G Marsden, and Mathieu Desbrun. On the geometric character of stress in continuum mechanics. *Zeitschrift für angewandte Mathematik und Physik*, 58(5):843–856, 2007. 151, 157

[67] George Karypis, Kirk Schloegel, and Vipin Kumar. ParMETIS - Parallel graph partitioning and sparse matrix ordering. Version 3.1. Technical report, University of Minnesota, Department of Computer Science and Engineering, August 2003. 20, 95

[68] Axel Klawonn. Block-Triangular Preconditioners for Saddle Point Problems with a Penalty Term. *SIAM J. Sci. Comput.*, 19(1):172–184, January 1998. 131

[69] Axel Klawonn. FETI domain decomposition methods for second order partial differential equations. *GAMM Mitteilungen*, 59(11):319–341, 2006. 19

[70] Axel Klawonn and Oliver Rheinbach. A parallel implementation of Dual-Primal FETI methods for three dimensional linear elasticity using a transformation of basis. *SIAM J. Sci. Comput.*, 28(5):1886–1906, 2006. 22, 23, 25, 26, 88, 89, 95, 99, 100, 101, 106, 108, 110, 112, 114, 115, 117, 119, 120, 122, 124, 125, 127, 129, 130

[71] Axel Klawonn and Oliver Rheinbach. Inexact FETI-DP methods. *Internat. J. Numer. Methods Engrg.*, 69:284–307, 2007. 19, 25, 131

[72] Axel Klawonn and Oliver Rheinbach. Robust FETI-DP methods for heterogeneous three dimensional elasticity problems. *Comput. Methods Appl. Mech. Engrg.*, 196(8):1400–1414, 2007. doi: 10.1016/j.cma.2006.03.023. 23, 24

[73] Axel Klawonn and Oliver Rheinbach. Highly scalable parallel domain decomposition methods with an application to biomechanics. *ZAMM Z. Angew. Math. Mech.*, 90(1):5–32, 2010. ISSN 0044-2267. doi: 10.1002/zamm.200900329. URL `http://dx.doi.org/10.1002/zamm.200900329`. i, 2, 3, 19, 25, 28, 88, 89, 90, 170

[74] Axel Klawonn and Olof B. Widlund. FETI and Neumann–Neumann iterative substructuring methods: Connections and new results. *Comm. Pure Appl. Math.*, 54:57–90, January 2001. 19

[75] Axel Klawonn and Olof B. Widlund. Dual and Dual–Primal FETI Methods for Elliptic Problems with Discontinuous Coefficients in Three Dimensions. In Tony Chan, Takashi Kako, Hideo Kawarada, and Olivier Pironneau, editors, *Domain Decomposition Methods in Sciences and Engineering: The Twelfth international Conference on Domain Decomposition Methods, held in Chiba, Japan, October 25–29, 1999*, pages 28–40. ddm.org, 2001. 19, 22

[76] Axel Klawonn and Olof B. Widlund. Selecting constraints in Dual-Primal FETI methods for elasticity in three dimensions. In R. Kornhuber, R.H.W. Hoppe, D.E. Keyes, J. Périaux, O. Pironneau, O. Widlund, and J. Xu, editors, *Domain Decomposition Methods in Science and Engineering*, pages 67–81. Springer-Verlag, Lecture Notes in Computational Science and Engineering, 2005. 22, 131

[77] Axel Klawonn and Olof B. Widlund. Dual-Primal FETI Methods for Linear Elasticity. *Comm. Pure Appl. Math.*, 59:1523–1572, 2006. 19, 22, 23, 24, 26

[78] Axel Klawonn, Patrick Radtke, and Oliver Rheinbach. FETI-DP with an Adaptive Coarse Space. *submitted October 2013, revised July 2014.* 28, 144

[79] Axel Klawonn, Olof B. Widlund, and Maksymilian Dryja. Dual-Primal FETI methods for three-dimensional elliptic problems with heterogeneous coefficients. *SIAM J. Numer. Anal.*, 40, 159-179 2002. 19, 22, 23, 24, 26, 89, 95, 99, 100, 101, 106

[80] Axel Klawonn, Olof B. Widlund, and Maksymilian Dryja. Dual-Primal FETI methods with face constraints. In Luca F. Pavarino and Andrea Toselli, editors, *Recent developments in domain decomposition methods*, pages 27–40. Springer-Verlag, Lecture Notes in Computational Science and Engineering, Volume 23, 2002. 19

[81] Axel Klawonn, Oliver Rheinbach, and Olof B. Widlund. Some computational results for Dual-Primal FETI methods for three dimensional elliptic problems. In R. Kornhuber, R.H.W. Hoppe, D.E. Keyes, J. Périaux, O. Pironneau, and J. Xu, editors, *Domain Decomposition Methods in Science and Engineering*, pages 361–368. Springer-Verlag, Lecture Notes in Computational Science and Engineering, 2005. Proceedings of the 15th International Conference on Domain Decomposition Methods, Berlin, July 21-25, 2003. 24

[82] Axel Klawonn, Oliver Rheinbach, and Barbara Wohlmuth. Dual-primal iterative substructuring for almost incompressible elasticity. In D.E. Keyes and O.B. Widlund, editors, *Domain Decomposition Methods in Science and Engineering*, volume 55, pages 399–406. Springer-Verlag, Lecture Notes in Computational Science and Engineering, 2006. 133

[83] Axel Klawonn, Luca F. Pavarino, and Oliver Rheinbach. Spectral element FETI-DP and BDDC preconditioners with multi-element subdomains. *Comput. Meth. Appl. Mech. Engrg.*, 198:511–523, 2008. 19

[84] Axel Klawonn, Oliver Rheinbach, and Olof B. Widlund. An analysis of a FETI–DP algorithm on irregular subdomains in the plane. *SIAM Journal on Numerical Analysis*, 46(5):2484–2504, 2008. doi: 10.1137/070688675. URL `http://link.aip.org/link/?SNA/46/2484/1`. 24

[85] Axel Klawonn, Martin Lanser, Patrick Radtke, and Oliver Rheinbach. On an Adaptive Coarse Space and on Nonlinear Domain Decomposition. In Jocelyne Erhel, Martin J. Gander, Laurence Halpern, Geraldine Pichot, Taoufik Sassi, and Olof B. Widlund, editors, *Proceedings of the 21st International Conference on Domain Decomposition Methods*, pages 49–56. DDM.org, 2012. 28, 144

[86] Axel Klawonn, Martin Lanser, and Oliver Rheinbach. Nonlinear FETI-DP and BDDC methods. *SIAM Journal on Scientific Computing*, 36(2):A737–A765, 2014. 28, 144

[87] Dana A. Knoll and David E. Keyes. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193(2):357 – 397, 2004. ISSN 0021-9991. doi: http://dx.doi.org/10.1016/j.jcp.2003.08.010. URL `http://www.sciencedirect.com/science/article/pii/S0021999103004340`. 27

[88] Jing Li and Xuemin Tu. Convergence analysis of a balancing domain decomposition method for solving a class of indefinite linear systems. *Numerical Linear Algebra with Applications*, 16(9):745–773, 2009. ISSN 1099-1506. doi: 10.1002/nla.639. URL `http://dx.doi.org/10.1002/nla.639`. 27

[89] Jing Li and Olof B. Widlund. FETI–DP, BDDC, and Block Cholesky Methods. *Internat. J. Numer. Methods Engrg.*, 66(2):250–271, 2006. 19, 133

[90] Jing Li and Olof B. Widlund. On the use of inexact subdomain solvers for BDDC algorithms. *Comput. Meth. Appl. Mech. Engrg.*, 196:1415–1428, 2007. 19

[91] The LifeV Team. LifeV, March 2014. `http://www.lifev.org/`. iii

[92] Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. *FFC: the FEniCS Form Compiler*, chapter 11. Springer, 2012. 45

[93] Jan Mandel and Clark R. Dohrmann. Convergence of a balancing domain decomposition by constraints and energy minimization. *Numer. Linear Algebra Appl.*, 10:639–659, 2003. 19

[94] Jan Mandel and Radek Tezaur. Convergence of a Substructuring Method with Lagrange Multipliers. *Numer. Math.*, 73:473–487, 1996. 19

[95] Jan Mandel and Radek Tezaur. On the convergence of a dual-primal substructuring method. *Numer. Math.*, 88:543–558, 2001. 19, 24, 26

[96] Jan Mandel, Clark R. Dohrmann, and Radek Tezaur. An algebraic theory for primal and dual substructuring methods by constraints. *Appl. Numer. Math.*, 54:167–193, 2005. 19

[97] Jerrold E. Marsden and Thomas J. R. Hughes. *Mathematical Foundations of Elasticity*. Dover Civil and Mechanical Engineering. Dover Publications, 1983. ISBN 9780486142272. 148, 151

[98] Jose Merodio and Patrizio Neff. A note on tensile instabilities and loss of ellipticity for a fiber-reinforced nonlinearly elastic solid. *Archives of Mechanics*, 58(3):293–303, 2006. 8, 163

[99] Joop C. Nagtegaal and David D. Fox. Using assumed enhanced strain elements for large compressive deformation. *International Journal of Solids and Structures*, 33(20):3151–3159, 1996. 16

[100] Patrizio Neff. A finite-strain elastic–plastic Cosserat theory for polycrystals with grain rotations. *International journal of engineering science*, 44(8):574–594, 2006. 154

[101] Patrizio Neff and Samuel Forest. A geometrically exact micromorphic model for elastic metallic foams accounting for affine microstructure. Modelling, existence of minimizers, identification of moduli and computational results. *Journal of Elasticity*, 87(2-3):239–276, 2007. 154

[102] Patrizio Neff, Andreas Fischle, and Ingo Münch. Symmetric Cauchy stresses do not imply symmetric Biot strains in weak formulations of isotropic hyperelasticity with rotational degrees of freedom. *Acta Mechanica*, 197(1-2):19–30, 2008. 159, 171

[103] Patrizio Neff, Jena Jeong, and Andreas Fischle. Stable identification of linear isotropic Cosserat parameters: bounded stiffness in bending and torsion implies conformal invariance of curvature. *Acta Mechanica*, 211(3-4):237–249, 2010. 171

[104] Patrizio Neff, Yuji Nakatsukasa, and Andreas Fischle. The unitary polar factor $Q = U$ minimizes norm $||\mathrm{Log}(Q^*Z)||^2$ and $||\mathrm{sym}\,\mathrm{Log}(Q^*Z)||^2$ in the spectral norm in any dimension and the Frobenius matrix norm in three dimensions. *arXiv preprint arXiv:1302.3235 to appear in SIAM Journal on Matrix Analysis and Applications (SIMAX)*, 2013. 171

[105] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer-Verlag, New York, 1999. 33, 34

[106] World Health Organization and others. Annex table 2: Deaths by cause, sex and mortality stratum in WHO regions, estimates for 2002. *The world health report*, 2004. ii

[107] Kendall H. Pierson. *A family of domain decomposition methods for the massively parallel solution of computational mechanics problems*. PhD thesis, University of Colorado at Boulder, Aerospace Engineering, 2000. 15, 19, 27

[108] Alfio Quarteroni and Alberto Valli. *Domain Decomposition Methods for Partial Differential Equations*. Oxford Science Publications, 1999. 19

[109] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics*, volume 37 of *Texts in Applied Mathematics*. Springer New York, 2007. 33

[110] Oliver Rheinbach. *Parallel Scalable Iterative Substructuring: Robust Exact and Inexact FETI-DP Methods with Applications to Elasticity*. PhD thesis, Fakultät für Mathematik, Universität Duisburg-Essen, Essen, Germany, 2006. ii, iii, iv, 15, 16, 22, 23, 36, 39, 80, 88, 95, 99, 100, 101, 106, 108, 110, 112, 114, 115, 117, 119, 120, 122, 124, 125, 127, 129, 130

[111] Giovanni Romano. Potential Operators and Conservative Systems. *Meccanica*, 7(3):141–146, 1972. 155

[112] Thomas Rumpel and Karl Schweizerhof. Hydrostatic Fluid Loading in Non-Linear Finite Element Analysis. *International Journal for Numerical Methods in Engineering*, 59(6):849–870, 2004. 155

[113] Jörg Schröder. Linear extrapolation of the displacement. Personal Communication, 2012. 31, 32, 137

[114] Jörg Schröder and Patrizio Neff. Invariant formulation of hyperelastic transverse isotropy based on polyconvex free energy functions. *International Journal of Solids and Structures*, 40:401–445, 2003. 3, 163

[115] Jörg Schröder and Patrizio Neff, editors. *Poly-, Quasi-and Rank-one Convexity in Applied Mechanics*. Number 516 in CISM International Centre for Mechanical Sciences. Springer, 2010. 162

[116] Jörg Schröder, Daniel Balzani, and Dietmar Gross. Aspects of modeling and computer simulation of soft tissues: Applications to arterial walls. *Materialwissenschaft und Werkstofftechnik*, 36(12):795–801, 2005. 2

[117] Jörg Schröder, Patrizio Neff, and Daniel Balzani. A variational approach for materially stable anisotropic hyperelasticity. *International Journal of Solids and Structures*, 42(15):4352–4371, 2005. 3

[118] Jörg Schröder, Axel Klawonn, Daniel Balzani, Oliver Rheinbach, and Dominik Brands. On mechanical modeling of arterial walls and parallel solution strategies. In Wolfgang Ehlers and Nils Karajan, editors, *Proceedings of the 2nd GAMM Seminar on Continuum Biomechanics*, 2007. Report no II-16 of the Institut für Mechanik (Bauwesen), Lehrstuhl II, Universität Stuttgart, Germany, 97-107. 2

[119] Hermann A. Schwarz. Über einen Grenzübergang durch alternierendes Verfahren. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 15:272–286, May 1870. 19

[120] Michael J. Sewell. On Configuration-Dependent Loading. *Archive for Rational Mechanics and Analysis*, 23(5):327–351, 1967. 155

[121] Juan C. Simo. The (Symmetric) Hessian for Geometrically Nonlinear Models in Solid Mechanics: Intrinsic Definition and Geometric Interpretation. *Computer Methods in Applied Mechanics and Engineering*, 96(2):189–200, 1992. 15

[122] Juan C. Simo. Numerical Analysis and Simulation of Plasticity. In Phillipe G. Ciarlet and Jacques L. Lions, editors, *Handbook of Numerical Analysis*, number 6. Elsevier Science, 1998. 16, 28, 90, 170

[123] Juan C. Simo, Robert L. Taylor, and Peter Wriggers. A Note on Finite-Element Implementation of Pressure Boundary Loading. *Communications in Applied Numerical Methods*, 7(7): 513–525, 1991. 155

[124] Barry F. Smith, Petter E. Bjørstad, and William Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996. 19

[125] Detlev Stalling, Malte Westerhoff, Hans-Christian Hege, et al. Amira: A highly interactive system for visual data analysis. *The visualization handbook*, 38:749–67, 2005. 93

[126] Richard L. Taylor and the FEAP team. FEAP Example Manual: v8.2. Technical report, UC Berkeley, . 40, 41

[127] Richard L. Taylor and the FEAP team. FEAP Programmer Manual: v8.2. Technical report, UC Berkeley, . 40, 42

[128] Richard L. Taylor and the FEAP team. FEAP Theory Manual: v8.2. Technical report, UC Berkeley, . 40, 169

[129] Richard L. Taylor and the FEAP team. FEAP User Manual: v8.2. Technical report, UC Berkeley, . 37, 40

[130] Richard L. Taylor and the FEAP team. The FEAP homepage, 2013. `http://www.ce.berkeley.edu/feap`. ii, 37

[131] John M. T. Thompson and Giles W. Hunt. *Elastic Instability Phenomena*. Wiley Chichester, 1984. 16

[132] Andrea Toselli and Olof B. Widlund. *Domain Decomposition Methods - Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin Heidelberg New York, 2005. 19, 20, 22, 27

[133] Paolo Tricerri, Luca Dedè, Alfio Quarteroni, and Adélia Sequeira. Numerical validation of isotropic and transversely isotropic constitutive models for healthy and unhealthy cerebral arterial tissues. MATHICSE Technical Report 39, École Polytechnique Fédérale de Lausanne, 2013. iv, 143

[134] Xuemin Tu. Three-level BDDC in two dimensions. *Inter. J. Numer. Methods Engrg.*, 69: 33–59, 2007. 19

[135] Xuemin Tu. Three-level BDDC in Three Dimensions. *SIAM J. Sci. Comput.*, 29(4):1759–1780 (electronic), 2007. ISSN 1064-8275. 19

[136] Julien Vanegue, Thomas Garnier, Julio Auto, Sebastien Roy, and Rafal Lesniak. Next generation debuggers for reverse engineering. *Black Hat Europe, Amsterdam, Netherlands*, page 1, 2007. 79

[137] Peter Wriggers. *Nichtlineare Finite-Element-Methoden*. Springer-Verlag Berlin Heidelberg, 2001. 6, 13, 16, 18, 27, 33, 148, 155