Scalable Space-Time Adaptive Simulation Tools for Computational Electrocardiology

Doctoral Dissertation submitted to the
Faculty of Informatics of the Università della Svizzera italiana
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Dorian Krause

under the supervision of
Prof. Rolf Krause

October 2013

Dissertation Committee

Prof. Rolf Krause	Università della Svizzera italiana, Switzerland
Prof. Illia Horenko	Università della Svizzera italiana, Switzerland
Prof. Igor Pivkin	Università della Svizzera italiana, Switzerland
Prof. Mark Potse	Università della Svizzera italiana, Switzerland
Prof. Luca F. Pavarino	Università degli Studi di Milano, Italy
Prof. Thomas Schulthess	Eidgenössische Technische Hochschule Zürich, Switzerland

Dissertation accepted on 4 October 2013

Research Advisor	PhD Program Director
Prof. Rolf Krause	**Prof. Antonio Carzaniga**

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Dorian Krause
Lugano, 4 October 2013

Abstract

This work is concerned with the development of computational tools for the solution of reaction-diffusion equations from the field of computational electrocardiology. We designed lightweight spatially and space-time adaptive schemes for large-scale parallel simulations.

We propose two different adaptive schemes based on locally structured meshes, managed either via a conforming coarse tessellation or a forest of shallow trees. A crucial ingredient of our approach is a non-conforming mortar element discretization which is used to glue together individually structured meshes by means of constraints. For the solution of variational problems in the proposed trial spaces we investigate two diametrically opposite approaches. First, we discuss the implementation of a matrix-free scheme for the solution of the monodomain equation on patch-wise adaptive meshes. Second, an approach to the construction of standard linear algebra data structures on tree-based meshes is considered. In particular, we address the element-wise assembly of stiffness matrices on constrained spaces via an algebraic representation of the inclusion map. We evaluate the performance of our adaptive schemes for small- and large-scale problems and demonstrate their applicability to the design of realistic large-scale heart models.

In order to enable local time stepping in the context of (semi-)implicit integration schemes, we present a space-time discretization based on the proposed lightweight adaptive mesh data structures. By means of a discontinuous Galerkin method in time, the solution of the linear or non-linear system of equations is reduced to a sequence of smaller systems of adjustable size. We discuss the stabilization of the arising discrete problems and present extensive numerical evaluations of the space-time adaptive solution of the $(1+1)$-, $(2+1)$- and $(3+1)$-dimensional heat equation as well as the monodomain reaction-diffusion equation. Our results show both feasibility and potential of adaptive space-time discretizations for the solution of reaction-diffusion equations in computational electrocardiology.

Acknowledgements

First and foremost I want to thank my advisor Prof. Rolf Krause for the support over the last years, for the patient supervision and for providing me with the opportunity to contribute to the development of the PROPAG code which was the starting point of this thesis work. The possibility to present my work at various international conferences is greatly appreciated.

I am thankful to Prof. Mark Potse for the many interesting and insightful conversations and the close cooperation.

I also want to express my gratitude to the other members of my dissertation committee, Prof. Illia Horenko, Prof. Luca Pavarino, Prof. Igor Pivkin and Prof. Thomas Schulthess, for their time and interest.

I am grateful to my fellow colleagues for their help at countless occasions and for the friendly working atmosphere.

Thanks to Dr. Daniel Ruprecht, Dr. Robert Speck and Dr. Thomas Dickopf for finding time to proofread parts of this work despite their busy schedule.

Last but not least I want to thank my family for their support and my parents for encouraging and actively backing my intellectual pursuits.

This work was partially funded by the "Swiss Platform for High-Performance and High-Productivity Computing" (HP2C) and profited from the funding for the project "A High Performance Approach to Cardiac Resynchronization Therapy" within the context of the "Iniziativa Ticino in Rete". Computational resources were generously provided by the University of Lugano, grants by the Swiss National Supercomputing Centre (CSCS) under the project IDs 268 and 397 as well as a preparatory project grant by PRACE.

This thesis was typeset with LuaTeX, Version beta-0.70.2-2012052410 (TeX Live 2012). Plots were prepared using METAPOST 1.504 (TeX Live 2012), PARAVIEW[76] 3.98.1 and MATPLOTLIB[87] 1.2.1.

Contents

Figures

Tables

Algorithms

1 Introduction

The study of the electrophysiology of the human heart is an important field in modern medicine and life sciences. As in most branches of science nowadays, computational modeling plays an important and increasingly pervasive role in electrophysiological studies [126]. In order to support these efforts, the community of computational mathematicians and computer scientists faces the challenge of developing computational tools for use by the domain scientists. Due to the fast-paced changes in computational hardware, these tools and the underlying methods and techniques need to be adapted or re-designed continuously for optimal performance.

This thesis is concerned with *efficient (space-time) adaptive tools for computational electrocardiology targeted at current and next-generation supercomputing systems*. We designed, implemented and experimentally evaluated novel adaptive schemes for the solution of non-linear reaction-diffusion equations. The research hypothesis underlying the presented work was that *non-conforming discretizations provide an excellent framework for the design of scalable adaptive algorithms based on lightweight data structures*.

In the first part of this thesis (Chapters 4 and 5) we consider spatially adaptive techniques with a focus on the monodomain equation [97]. The monodomain equation is a non-linear reaction-diffusion equation used extensively in computational electrocardiology (see, for example, Potse et al. [127]). In practice, this equation is often solved using low-order, semi-implicit time discretization schemes that can be implemented very efficiently due to a weak diffusion term. Therefore we expect current state-of-the-art parallel adaptive techniques (see, for example, Burstedde et al. [35]), which were developed for strongly non-linear and ill-conditioned problems, to be unsuited for our use case.

Instead, we pay particular attention to the cost per degree of freedom and the underlying mesh data structures. Non-conforming discretization techniques, in particular the *mortar element method* [25] that is used throughout this thesis, allow for the flexible construction of adaptive meshes (or, to be more precise, approximation spaces) by "gluing" together local pieces. Our basic building blocks are structured/tensor meshes, a data structure that is equally well suited for current latency-optimized processing units (such as standard x86 central processing units) and for throughput-optimized processing units (such as graphical processing units).

In this thesis we propose two different adaptive schemes based on locally structured meshes, managed either via a conforming tessellation (Chapter 4) or a forest of shallow trees (Chapter 5).

These mesh data structures are characterized by their low memory footprint. We present two diametrically opposite approaches to the design of the parallelized algebra data structures. On the one hand, we propose a matrix-free implementation that allows us to fully exploit the special mesh structure but is limited to the solution of reaction-diffusion equations using semi-implicit time stepping and block preconditioning. On the other hand, we discuss an approach based on standard linear algebra data structures that cannot take advantage of the local structure of the non-conforming meshes but are flexible and can be combined with a variety of preconditioning techniques.

We assess the performance of our solution schemes in several numerical experiments and demonstrate the applicability of the proposed adaptive techniques for the design of realistic large-scale heart models.

In the second part of this thesis (Chapter 6) we discuss combined space-time adaptivity. In many cases of interest, global time step control is inefficient because the time step is globally adjusted to the local features of the solution [166]. Local time stepping [67] on the other hand is not easily combined with implicit or semi-implicit time discretizations. We consider space-time discretizations as a means to enable local time stepping in the context of (semi-)implicit discretizations.

We employ a hybrid space-time discretization that combines non-conforming finite elements within a space-time slab with a discontinuous Galerkin method [92] in time in order to decouple individual space-time slabs. This discretization scheme allows us to reuse the adaptive mesh data structures and discretization schemes developed in Chapters 4 and 5 for a combined space-time adaptive solution scheme. Since we employ quadrilateral or hexahedral tessellations and local tensor meshes, our mesh data structures naturally generalize to arbitrary dimensions. Our long-term goal is the space-time adaptive solution of $(3+1)$-dimensional large-scale problems. The lightweight nature of the employed mesh data structures is crucial for the feasibility of such simulations on supercomputers with their limited amounted of main memory per core.

We present extensive numerical experiments that prove the feasibility of our approach and highlight challenges that need to be addressed in future work.

Contributions

Our work contributes, on the one hand, to the on-going exploration of the design space of adaptive methods on contemporary high performance platforms and, on the other hand, to research efforts on fast solution techniques for computational electrocardiology. We present adaptive strategies that combine the performance advantages of structured meshes with the flexibility of non-conforming mortar discretizations in a novel and original fashion. These methods can be used for both space and space-time adaptive simulations of non-linear reaction-diffusion equations.

In this thesis we take a holistic approach to the design of adaptive solution schemes that combines the design of the mesh data structures, the definition of appropriate ansatz spaces as well as considerations about implementation and parallelization. In addition, we present insightful numerical experiments to assess the performance of our designs.

Outline

This thesis is organized as follows. In Chapter 2 we introduce the governing equations used for modeling the electrical properties of cells and tissue. We review discretization techniques for these equations and discuss the spatial discretization using a symmetric Galerkin method as well as different low-order time discretizations. Finally, we motivate the study of adaptive techniques for the solution of the bidomain and monodomain equations and review the state of the art in this field.

In Chapter 3 we discuss the hybrid parallelization of the PROPAG heart model. This chapter serves two purposes. On the one hand we present a state-of-the-art computational heart model on uniform meshes and thus show the performance level that our adaptive schemes are to compete with. On the other hand we present a performance analysis of the new hybrid OpenMP+MPI parallelization in PROPAG-5, which is of interest in its own right.

In Chapter 4 we present a lightweight adaptive discretization scheme for the monodomain equation. We introduce the mortar element method in the context of a geometrically conforming tessellation and propose a matrix-free implementation. Numerical experiments are discussed and a comparison with related work is drawn.

In Chapter 5 we present an alternative scheme based on forests of shallow trees. This design allows for a finer control over the refined regions compared to the lightweight scheme from Chapter 4. We discuss the construction of approximation spaces and the assembly of mass and stiffness matrices on these meshes and present numerical experiments to assess the performance of this approach.

In Chapter 6 we discuss the extension of our previous work to space-time adaptivity by means of a hybrid finite element/discontinuous Galerkin space-time discretization. We present extensive numerical experiments that show the effectiveness of space-time adaptive discretizations and demonstrate the feasibility of the approach, even for $(3+1)$-dimensional problems.

2 Computational Modeling in Electrophysiology

In this chapter we introduce the governing equations used to simulate the activation sequence of the human heart. The focus of our presentation will be on the mathematical aspects. For more details on the physiological background we refer the reader to the books by Keener and Sneyd [96,97] on which the following introduction is largely based.

We start by discussing models for the ionic current through cell membranes by looking at single cells. In particular, we introduce the membrane model developed by Bernus et al. [27] which we use in most of our numerical studies. We then introduce the bidomain and monodomain equations for modeling cardiac tissue and discuss numerical methods for the solution of these equations. Finally, we motivate the use of adaptive techniques for this problem class and review the existing literature.

2.1 Modeling Electrical Properties of Cardiac Cells

Cells maintain an ion concentration difference between the interior and exterior of the cell by means of active pumps (such as the Na^+-K^+ ATPase pumps [96]). In consequence, a difference between the intra-cellular potential φ_i and the extra-cellular potential φ_e exists. By convention the *membrane voltage* V equals $\varphi_i - \varphi_e$ and is usually measured in mV.

The cell membrane can be considered an insulator with capacitance C_m, i.e.,

$$C_m \cdot V = \Delta Q \tag{2.1}$$

where ΔQ denotes the charge difference between the intra- and extra-cellular domain. For our purposes $C_m = 1\ \mu\mathrm{F/cm^2}$. In the cell membrane of excitable cells, millions of ion channels are embedded which actively transport ions through the cell membrane upon activation. This creates a current

$$I_{ion} = -\frac{d\Delta Q}{dt}\ . \tag{2.2}$$

Since the activity of ion channels is steered by the membrane potential V, the ionic current I_{ion} depends on the membrane potential. Combining equations (2.1) and (2.2) we find that the capacitive and ionic current balance each other, i.e.,

$$C_m \frac{dV}{dt} + I_{ion}(V) = 0\ . \tag{2.3}$$

From a modeling point of view, the challenge is to derive an analytic expression for the dependence of I_{ion} on the membrane voltage and potentially other variables that model the state of ion channels or time-dependent ionic concentrations. The current generated by the transport of ions of type S is often expressed as

$$I_{\text{S}} = g_{\text{S}} \, (V - V_{\text{S}})$$

with the conductance g_{S} and the constant Nernst potential V_{S}. Note that for a given ion type S, both inward and outward currents might contribute to I_{ion}. As g_{S} depends on the state of the ion channels that pump ions in or out of the cell, the value of g_{S} will be time-dependent and be implicitly coupled to the membrane voltage V.

2.1.1 Hodgkin-Huxley Type Models

In 1952 Hodgkin and Huxley [78] proposed a model for I_{ion} for giant squid axons. This work had profound impact on many branches of physiology, earning them a Nobel prize in physiology or medicine in 1963. Despite its inadequacy for the modeling of cardiac cells we shortly discuss the model because of its profound impact on the development of membrane models. The Hodgkin-Huxley equations state that

$$I_{\text{ion}}(V,n,m,h) = \overline{g}_{\text{K}} n^4 \, (V - V_{\text{K}}) + \overline{g}_{\text{Na}} m^3 h \, (V - V_{\text{Na}}) + \overline{g}_{\text{L}} \, (V - V_{\text{L}}) \tag{2.4}$$

where the *gating variables* (n,m,h) obey linear differential equations with voltage-dependent steady states and relaxation times. More precisely, each gating variable $u \in \{n,m,h\}$ obeys the equation

$$\dot{u} = \frac{u_\infty(V) - u}{\tau_u(V)} \, . \tag{2.5}$$

For constant membrane voltage V on the time interval $(0,t)$, this equation is solved by

$$u(t) = u_\infty(V) - \big(u_\infty(V) - u(0)\big) \mathrm{e}^{-t/\tau_u(V)} \, . \tag{2.6}$$

Note that equation (2.5) can be reformulated as

$$\dot{u} = \alpha_u(V) \, (1 - u) + \beta_u(V) \, u \tag{2.7}$$

with

$$u_\infty(V) = \frac{\alpha_u(V)}{\alpha_u(V) + \beta_u(V)} \quad \text{and} \quad \tau_u(V) = \frac{1}{\alpha_u(V) + \beta_u(V)} \, .$$

The variables $\alpha_u(V)$, $\beta_u(V)$ can be interpreted as rates of the opening and closing of ion channel gates.

The variable n controls the activation and deactivation of potassium channels. The activation and deactivation of sodium channels is controlled by m and h, respectively. Because the gating variables m and h have different kinetics (m being a *fast variable* and h a *slow variable*) the experimentally measured sodium conductance cannot be modeled with a single gating variable.

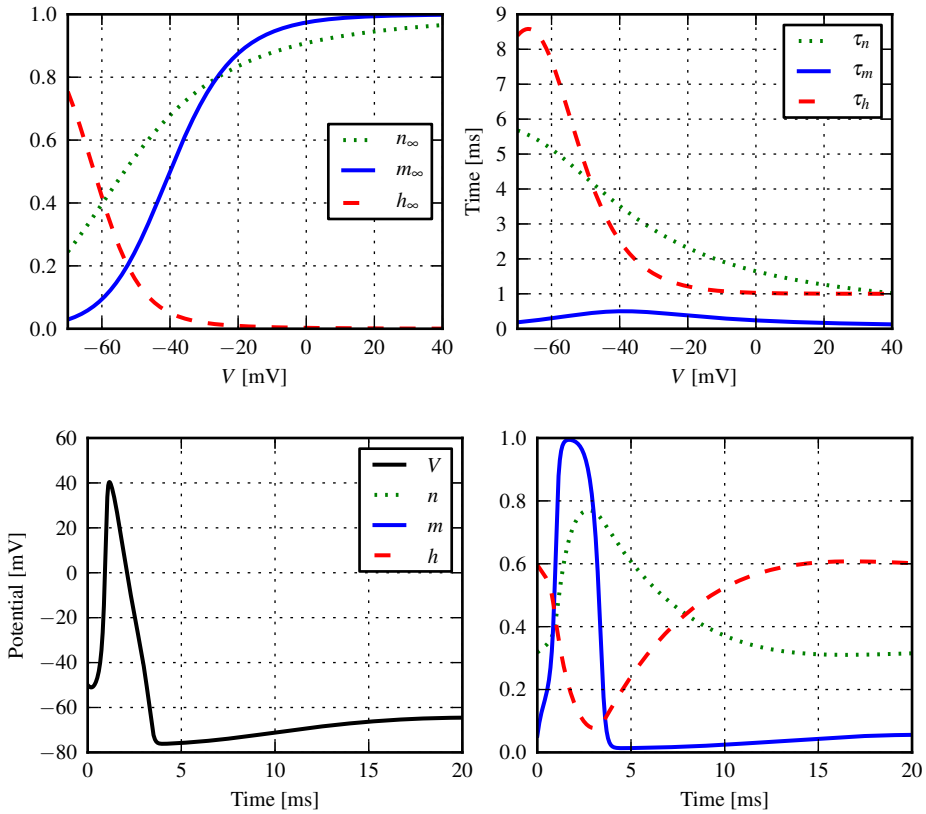


Figure 2.1. *Behavior of the Hodgkin-Huxley model. The upper plots show the dependency of the steady-state values and relaxation times on V. The lower plots show the solution of equation* (2.3) *with an initial voltage difference of +15 mV relative to the equilibrium value of -65 mV.*

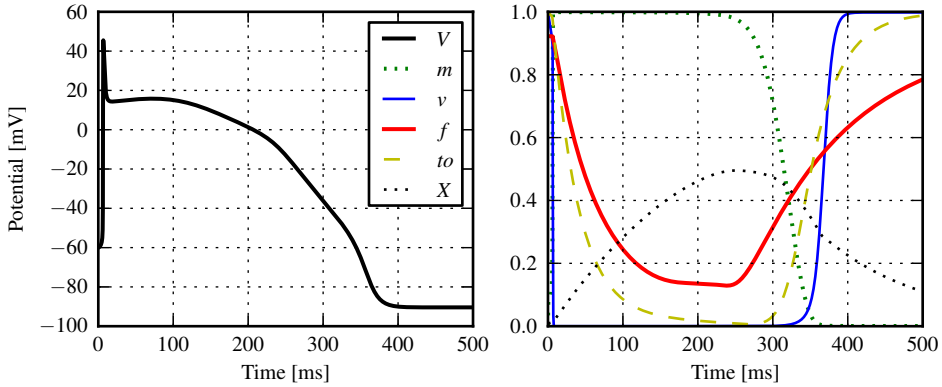


Figure 2.2. *Solution of equation* (2.3) *with an initial voltage difference of +30.272 mV (relative to the equilibrium value of -90.272 mV) using the Bernus membrane model.*

In the upper plot in Figure 2.1 the voltage dependency of n_∞, m_∞, h_∞ and τ_n, τ_m, τ_h is shown. The membrane voltage and gating variable values created by an initial clamped membrane voltage is shown in the lower part of Figure 2.1. Parameters for $\overline{g}_{\mathrm{K}}$, V_{K}, $\overline{g}_{\mathrm{Na}}$, V_{Na}, $\overline{g}_{\mathrm{L}}$ and V_{L} were taken from Keener and Sneyd [96].

Equation (2.4) itself is not appropriate for modeling the ionic current in cardiac cells. However, in many models for I_{ion} the conductance is expressed as a monomial in the state vector $\mathbf{s} \in \mathbb{R}^S$ similar to the Hodgkin-Huxley model.

2.1.2 Membrane Models for Human Ventricular Cells

A large number of membrane models with different complexities and different applicability are available in the literature [42]. A first generic model for mammalian ventricular cells was published by Beeler and Reuter [15]. A generalized version was published by Luo and Rudy [106] which used more recent experimental information from guinea pigs. This membrane model is known as the *phase-1 Luo-Rudy* model. In 1994, Luo and Rudy [107,108] published an improved version of their membrane model which is known as the *phase-2 Luo-Rudy* model. Priebe and Beuckelmann [129] adapted the phase-2 Luo-Rudy model using human data. In 2002, Bernus et al. [27] developed a reduced version of the Priebe-Beuckelmann model to lower the computational cost. We will discuss the Bernus membrane model in Section 2.1.4. A different model for human ventricular cells was proposed by ten Tusscher et al. [151] in 2004 with an update in 2006, see Ref. 150.

The Priebe-Beuckelmann membrane model features nine gating variables and four time dependent ion concentrations. The reduced Bernus model features five gating variables and no ion concentrations. The model by ten Tusscher et al. requires integration of thirteen gating variables and four time-dependent ion concentrations. Newer models might feature even more state variables. For example, Iyer et al. [91] developed a membrane model, based on a Markov state approach, with a total of 65 state variables.

2.1.3 The Fitz-Hugh Nagumo Model

The Fitz-Hugh Nagumo model is a simplified model that is useful for testing new numerical methods. The Fitz-Hugh Nagumo model contains a single slow gating variable w. Several different versions of the model can be found in the literature. During early testing of the techniques developed in this thesis, we used the following version of the model:

$$\begin{aligned} I_{\mathrm{ion}} &= V \cdot (1 - (V/13)) \cdot (1 - (V/100)) + 4.4w \,, \\ \dot{w} &= 0.012 \cdot (V/100 - w) \,. \end{aligned}$$

2.1.4 The Bernus Model

In large parts of this work we will make use of the Bernus membrane model because it has moderate computational cost but is still able to produce realistic results [127].

As stated earlier, the Bernus membrane model is a reduced version of the Priebe-Beuckelmann model developed with the goal of faster two- and three-dimensional simulations of reentrant arrhythmia[27]. The model contains five state variables

$$\mathbf{s} = (m, v, f, to, X)$$

that regulate the fast Na^+ current (variables m and v), the slow Ca^{2+} current (variable f), the transient outward current (variable to) and the inward delayed rectifier K^+ current (variable X). The model features no time-dependent ionic concentrations. The ionic current equals

$$I_{\text{ion}} = I_{\text{Na}} + I_{\text{Ca}} + I_{\text{to}} + I_{\text{K}} + I_{\text{K1}} + I_{\text{Na,b}} + I_{\text{Ca,b}} + I_{\text{NaK}} + I_{\text{NaCa}}$$

with the fast Na^+ inward current I_{Na}, the slow Ca^{2+} inward current I_{Ca}, the transient outward current I_{to}, the outward delayed rectifier K^+ current I_{K}, the inward rectifier K^+ current I_{K1}, the Ca^{2+} and Na^+ background currents $I_{\text{Na,b}}$ and $I_{\text{Ca,b}}$, as well as the pump and exchange currents I_{NaK} and I_{NaCa}.

In Figure 2.2 the solution of equation (2.3) with initially clamped membrane voltage $V = -60\,\text{mV}$ is shown. This plot illustrates the differences in length scale between the fast depolarization and slow repolarization that is characteristic for cardiac myocytes.

2.2 Modeling Electrical Properties of Cardiac Tissue

The purpose of the electrical activation of the heart tissue is the initiation of a mechanical contraction in order to pump blood through the heart chambers. A coordinated contraction is a prerequisite for an efficient pumping functionality. The action potentials originate at the *sinoatrial node* from where they spread via cell-to-cell conduction[98]. The action potentials enter the ventricles through the *atrioventricular node* which is connected to the *bundle of His*, followed by the left and right *bundle branches* that end in a complicated network known as the *Purkinje system*. The conduction velocity in the Purkinje system is about eight times higher than the conduction velocity in the surrounding ventricular myocyte tissue that is excited through the connection to the Purkinje fibers. In order to accurately model the electrophysiology of the heart it is therefore important to study the propagation of the action potential through excitable tissue.

2.2.1 The Bidomain equation

The *bidomain equation*[158] is generally accepted as the governing equation for the electrical propagation in cardiac tissue[97]. It is based on a two-phase representation of the cardiac tissue, i.e., intra-cellular and extra-cellular domain occupy the same space. Assuming Ohmic materials, conservation of the total current (in absence of external currents) states

$$\boldsymbol{\nabla} \cdot (\mathbf{G}_{\text{i}} \boldsymbol{\nabla} \varphi_{\text{i}}) + \boldsymbol{\nabla} \cdot (\mathbf{G}_{\text{e}} \boldsymbol{\nabla} \varphi_{\text{e}}) = 0 \tag{2.8}$$

where $\mathbf{G}_{\text{i}}$ and $\mathbf{G}_{\text{e}}$ denote the conductivity tensors in the intra- and extra-cellular domain, respectively. The *membrane current* I_{m}, i.e., the sum of the capacitive and ionic current, equals the current

leaving the intra-cellular space up to a multiplicative factor χ, the *surface-to-volume ratio*. In the following we will always assume $\chi = 1000\ \mathrm{cm}^{-1}$. Hence,

$$\chi\left(C_\mathrm{m}\partial_t V + I_\mathrm{ion}(V)\right) = \boldsymbol{\nabla}\cdot\left(\mathbf{G}_\mathrm{i}\boldsymbol{\nabla}\varphi_\mathrm{i}\right) = -\boldsymbol{\nabla}\cdot\left(\mathbf{G}_\mathrm{e}\boldsymbol{\nabla}\varphi_\mathrm{e}\right) \tag{2.9}$$

Inserting $V = \varphi_\mathrm{i} - \varphi_\mathrm{e}$ we obtain the bidomain reaction-diffusion equation

$$\begin{aligned}\chi\left(C_\mathrm{m}\partial_t\varphi_\mathrm{i} - C_\mathrm{m}\partial_t\varphi_\mathrm{e} + I_\mathrm{ion}\left(\varphi_\mathrm{i}-\varphi_\mathrm{e}\right)\right) &= \boldsymbol{\nabla}\cdot\left(\mathbf{G}_\mathrm{i}\boldsymbol{\nabla}\varphi_\mathrm{i}\right)\ ,\\ \chi\left(C_\mathrm{m}\partial_t\varphi_\mathrm{e} - C_\mathrm{m}\partial_t\varphi_\mathrm{i} - I_\mathrm{ion}\left(\varphi_\mathrm{i}-\varphi_\mathrm{e}\right)\right) &= \boldsymbol{\nabla}\cdot\left(\mathbf{G}_\mathrm{e}\boldsymbol{\nabla}\varphi_\mathrm{e}\right)\ .\end{aligned} \tag{2.10}$$

This equation is known as the *parabolic-parabolic* formulation of the bidomain equation. An alternative formulation, known as the *parabolic-elliptic* formulation of the bidomain equation, is obtained as follows. By definition of V and equation (2.8) one finds that the extra-cellular potential and the membrane voltage are related by

$$\boldsymbol{\nabla}\cdot\left(\mathbf{G}_\mathrm{i}\boldsymbol{\nabla}V\right) + \boldsymbol{\nabla}\cdot\left(\left(\mathbf{G}_\mathrm{i}+\mathbf{G}_\mathrm{e}\right)\boldsymbol{\nabla}\varphi_\mathrm{e}\right) = 0\ . \tag{2.11}$$

Similarly, by inserting $\varphi_\mathrm{i} = V + \varphi_\mathrm{e}$ in equation (2.9) we find

$$\chi\left(C_\mathrm{m}\partial_t V + I_\mathrm{ion}(V)\right) = \boldsymbol{\nabla}\cdot\left(\mathbf{G}_\mathrm{i}\boldsymbol{\nabla}\left(V+\varphi_\mathrm{e}\right)\right)\ . \tag{2.12}$$

The coupled system constituted by equation (2.12) and equation (2.11) is known as the *parabolic-elliptic* formulation of the bidomain equation.

It is worth noting that a more rigorous derivation of the bidomain equation using mathematical homogenization techniques is possible, see, for example, Keener and Sneyd [97].

2.2.2 The Monodomain equation

An important simplification of the bidomain equation is the *monodomain equation*

$$\chi\left(C_\mathrm{m}\partial_t V + I_\mathrm{ion}(V)\right) = \boldsymbol{\nabla}\cdot\left(\mathbf{G}_\mathrm{mono}\boldsymbol{\nabla}V\right)\ , \tag{2.13}$$

where (component-wise)

$$\mathbf{G}_\mathrm{mono} = \frac{\mathbf{G}_\mathrm{i}\cdot\mathbf{G}_\mathrm{e}}{\mathbf{G}_\mathrm{i}+\mathbf{G}_\mathrm{e}}\ .$$

Formally, equation (2.13) is obtained from (2.8) and (2.9) by assuming the intra- and extra-cellular conductivity tensors to be linear dependent. Even though this assumption is usually not valid, monodomain simulations can approximate bidomain simulations well for large-scale models, see Bordas et al. [31], Potse et al. [127]. Note that it is possible to compute the extra-cellular (and therefore also the intra-cellular potential) by solving equation (2.11) with the membrane voltage V obtained as the solution of the monodomain equation.

2.2.3 Conductivity Tensors

The conductivity tensors $\mathbf{G}_\mathrm{i}$, $\mathbf{G}_\mathrm{e}$ and $\mathbf{G}_\mathrm{mono}$ are usually expressed as

$$\mathbf{G} = G_\mathrm{l}\,\mathbf{a}_\mathrm{l}\otimes\mathbf{a}_\mathrm{l} + G_\mathrm{t}\,\mathbf{a}_\mathrm{t}\otimes\mathbf{a}_\mathrm{t} + G_\mathrm{n}\,\mathbf{a}_\mathrm{n}\otimes\mathbf{a}_\mathrm{n}$$

in units of $\mathrm{mS/cm}$. The local orthonormal basis $(\mathbf{a}_\mathrm{l}(\mathbf{x}), \mathbf{a}_\mathrm{t}(\mathbf{x}), \mathbf{a}_\mathrm{n}(\mathbf{x}))$ describes the orientation of the fibers in the cardiac muscle. Usually, the diffusion coefficient G_l along the fibers is dominant.

2.2.4 Summary of Governing Equations

For the purpose of future referencing we now list the strong forms of the governing equations considered in this thesis. Let $\Omega \subset \mathbb{R}^d$ be a bounded domain and $(0,T)$ the time interval of interest. By $I_{\text{app}} : \Omega \times (0,T) \to \mathbb{R}$ we denoted the applied current. By $\mathbf{s}$ and $\mathbf{Z}$ we denote the vector of state variables of our membrane model and the right-hand sides of the governing ordinary differential equations for the state variables, respectively.

Bidomain equation (parabolic-parabolic). Find $(\varphi_\text{i}, \varphi_\text{e}) \in \mathcal{C}^1\left((0,T), \mathcal{C}^2(\Omega)\right)^2$ such that

$$\begin{aligned}
C_\text{m}\partial_t\varphi_\text{i} - C_\text{m}\partial_t\varphi_\text{e} &= \frac{1}{\chi}\nabla\cdot(\mathbf{G}_\text{i}\nabla\varphi_\text{i}) - I_\text{ion}(\varphi_\text{i}-\varphi_\text{e},\mathbf{s}) + I_\text{app} \\
C_\text{m}\partial_t\varphi_\text{e} - C_\text{m}\partial_t\varphi_\text{i} &= \frac{1}{\chi}\nabla\cdot(\mathbf{G}_\text{e}\nabla\varphi_\text{e}) + I_\text{ion}(\varphi_\text{i}-\varphi_\text{e},\mathbf{s}) - I_\text{app} \qquad \text{in} \quad \Omega\times(0,T)\,, \\
\partial_t\mathbf{s} &= \mathbf{Z}(\varphi_\text{i}-\varphi_\text{e},\mathbf{s}) \qquad \text{in} \quad \Omega\times(0,T)\,, \\
\mathbf{n}\cdot\mathbf{G}_\text{i}\nabla\varphi_\text{i} &= 0 \qquad \text{on} \quad \partial\Omega\times(0,T)\,, \\
\mathbf{n}\cdot\mathbf{G}_\text{e}\nabla\varphi_\text{e} &= 0 \qquad \text{on} \quad \partial\Omega\times(0,T)\,.
\end{aligned} \tag{2.14}$$

Bidomain equation (parabolic-elliptic). Find $(V,\varphi_\text{e}) \in \mathcal{C}^1\left((0,T),\mathcal{C}^2(\Omega)\right)\times\mathcal{C}^0\left((0,T),\mathcal{C}^2(\Omega)\right)$ such that

$$\begin{aligned}
C_\text{m}\partial_t V &= \frac{1}{\chi}\nabla\cdot(\mathbf{G}_\text{i}\nabla(V+\varphi_\text{e})) - I_\text{ion}(V,\mathbf{s}) + I_\text{app} \\
\nabla\cdot(\mathbf{G}_\text{i}\nabla V) + \nabla\cdot((\mathbf{G}_\text{i}+\mathbf{G}_\text{e})\nabla\varphi_\text{e}) &= 0 \qquad \text{in} \quad \Omega\times(0,T)\,, \\
\partial_t\mathbf{s} &= \mathbf{Z}(V,\mathbf{s}) \qquad \text{in} \quad \Omega\times(0,T)\,, \\
\mathbf{n}\cdot\mathbf{G}_\text{i}\nabla(V+\varphi_\text{e}) &= 0 \qquad \text{on} \quad \partial\Omega\times(0,T)\,, \\
\mathbf{n}\cdot\mathbf{G}_\text{i}\nabla V + \mathbf{n}\cdot(\mathbf{G}_\text{i}+\mathbf{G}_\text{e})\nabla\varphi_\text{e} &= 0 \qquad \text{on} \quad \partial\Omega\times(0,T)\,.
\end{aligned} \tag{2.15}$$

Monodomain equation. Find $V \in \mathcal{C}^1\left((0,T),\mathcal{C}^2(\Omega)\right)$ such that

$$\begin{aligned}
C_\text{m}\partial_t V &= \frac{1}{\chi}\nabla\cdot(\mathbf{G}_\text{mono}\nabla V) - I_\text{ion}(V,\mathbf{s}) + I_\text{app} \qquad \text{in} \quad \Omega\times(0,T)\,, \\
\partial_t\mathbf{s} &= \mathbf{Z}(V,\mathbf{s}) \qquad \text{in} \quad \Omega\times(0,T)\,, \\
\mathbf{n}\cdot\mathbf{G}_\text{mono}\nabla V &= 0 \qquad \text{on} \quad \partial\Omega\times(0,T)\,.
\end{aligned} \tag{2.16}$$

Note that no boundary conditions for $\mathbf{s}$ are enforced as the equations for $\mathbf{s}$ are spatially decoupled.

These equations are further augmented with appropriate initial conditions. In this work we usually use constant initial conditions with $V(0)$ equal to the rest potential and $\varphi_\text{e} = 0$. Note that, depending on the simulation, different boundary conditions are used in the literature [120].

The bidomain equation is a *degenerate* reaction-diffusion equation since the potentials φ_i, φ_e are only well defined up to constants, i.e., the equations are invariant under the transformation

$$\begin{aligned}
\varphi_\text{i}(\mathbf{x},t) &\leftarrow \varphi_\text{i}(\mathbf{x},t) - \beta(t)\,, \\
\varphi_\text{e}(\mathbf{x},t) &\leftarrow \varphi_\text{e}(\mathbf{x},t) - \beta(t)\,,
\end{aligned}$$

for $\beta \in C^1((0,T),\mathbb{R})$. A common approach to deal with the degenerate nature of the equations is to search for solutions with zero mean, i.e.,

$$\int_\Omega \varphi_\mathrm{i}(\mathbf{x},t)\,\mathrm{d}\mathbf{x} = \int_\Omega \varphi_\mathrm{e}(\mathbf{x},t)\,\mathrm{d}\mathbf{x} = 0 \quad \text{for all } t \in (0,T)\ .$$

2.3 Numerical Methods

In this section we review popular numerical schemes for the solution of the mono- and bidomain equations.

2.3.1 Spatial Discretization

In the literature, finite difference, finite volume and finite element methods have been used to discretize the bidomain or monodomain equations[162]. For realistic whole-heart simulations, an important requirement for the spatial discretization is the ability to cope with discontinuities in the conductivity values that result from the differences in tissue type.

Saleheen and Ng[138] proposed a finite difference method particularly for dealing with jumps in the conductivity tensors $\mathbf{G}_\mathrm{i}$, $\mathbf{G}_\mathrm{e}$ or $\mathbf{G}_\mathrm{mono}$. A realistic heart model using this discretization has been developed by Potse et al.[127]. Finite volume discretizations have been used, for example, by Harrild and Henriquez[73]. The most popular discretization scheme for the bidomain and monodomain equations, however, is the finite element method which is used in several computational models[28,46,113,156]. Finite elements (and, to some extend, finite volume) methods have the advantage of a flexible handling of unstructured meshes for complicated domains and can naturally cope with discontinuous conductivity values as long as the jumps are aligned with element faces. An important advantage of finite difference discretizations is that they naturally lead to uncoupled ordinary differential equations for the membrane state variables. In standard finite element methods, in contrast, the state variables are coupled via the non-vanishing off-diagonal entries in the mass matrix.

With the exception of Chapter 3, where we discuss the parallelization of the finite differences-based PROPAG model, the work presented in this thesis is focused on finite element discretizations. In the following, we shortly review the weak formulation of the bidomain and monodomain equations and the resulting coupled ordinary differential equations when using the method of lines.

The weak formulation of equations (2.14)–(2.16) is obtained by testing the equations with functions $U \in H^1(\Omega)$. By applying integration by parts to the diffusion terms, the regularity requirements for the solution can be reduced to the existence of a first weak derivative. Note that, by definition of the free Neumann boundary conditions in equations (2.14)–(2.16), the boundary integrals that results from applying the divergence theorem vanish. We consider the symmetric Galerkin approximation of the resulting equations using a conforming approximation space $\mathbb{Y} \subset H^1(\Omega)$ with basis $\boldsymbol{\pi} = \{\pi_\alpha\}$. In equations (2.17)–(2.19) below we state the weak formulation of the bidomain equation in parabolic-parabolic and parabolic-elliptic form, as well as the weak formulation of the monodomain equation.

Bidomain equation (parabolic-parabolic). Find $(\varphi_\mathrm{i}, \varphi_\mathrm{e}) \in \mathcal{C}^1\left((0,T), \mathbb{Y}/\mathbb{R}\right)^2$ such that

$$\begin{aligned}
(C_\mathrm{m}\partial_t\varphi_\mathrm{i} - C_\mathrm{m}\partial_t\varphi_\mathrm{e}, U)_{L^2(\Omega)} &= -\frac{1}{\chi}\left(\mathbf{G}_\mathrm{i}\boldsymbol{\nabla}\varphi_\mathrm{i}, \boldsymbol{\nabla}U\right)_{L^2(\Omega)} - \left(I_\mathrm{ion}(\varphi_\mathrm{i}-\varphi_\mathrm{e}, \mathbf{s}) - I_\mathrm{app}, U\right)_{L^2(\Omega)} \ , \\
(C_\mathrm{m}\partial_t\varphi_\mathrm{e} - C_\mathrm{m}\partial_t\varphi_\mathrm{i}, U)_{L^2(\Omega)} &= -\frac{1}{\chi}\left(\mathbf{G}_\mathrm{e}\boldsymbol{\nabla}\varphi_\mathrm{e}, \boldsymbol{\nabla}U\right)_{L^2(\Omega)} + \left(I_\mathrm{ion}(\varphi_\mathrm{i}-\varphi_\mathrm{e}, \mathbf{s}) - I_\mathrm{app}, U\right)_{L^2(\Omega)} \ , \\
(\partial_t\mathbf{s}, U)_{L^2(\Omega)} &= (\mathbf{Z}(\varphi_\mathrm{i}-\varphi_\mathrm{e}, \mathbf{s}), U)_{L^2(\Omega)} \ ,
\end{aligned} \tag{2.17}$$

for all $U \in \mathbb{Y}$.

Bidomain equation (parabolic-elliptic). Find $(V, \varphi_\mathrm{e}) \in \mathcal{C}^1\left((0,T), \mathbb{Y}\right) \times \mathcal{C}^0\left((0,T), \mathbb{Y}/\mathbb{R}\right)$ such that

$$\begin{aligned}
(C_\mathrm{m}\partial_t V, U)_{L^2(\Omega)} &= -\frac{1}{\chi}\left(\mathbf{G}_\mathrm{i}\boldsymbol{\nabla}\left(V+\varphi_\mathrm{e}\right), \boldsymbol{\nabla}U\right)_{L^2(\Omega)} - \left(I_\mathrm{ion}(V, \mathbf{s}) - I_\mathrm{app}, U\right)_{L^2(\Omega)} \ , \\
\left((\mathbf{G}_\mathrm{i}+\mathbf{G}_\mathrm{e})\boldsymbol{\nabla}\varphi_\mathrm{e}, \boldsymbol{\nabla}U\right)_{L^2(\Omega)} &= -\left(\mathbf{G}_\mathrm{i}\boldsymbol{\nabla}V, \boldsymbol{\nabla}U\right)_{L^2(\Omega)} \ , \\
(\partial_t\mathbf{s}, U)_{L^2(\Omega)} &= (\mathbf{Z}(V, \mathbf{s}), U)_{L^2(\Omega)} \ ,
\end{aligned} \tag{2.18}$$

for all $U \in \mathbb{Y}$.

Monodomain equation. Find $V \in \mathcal{C}^1\left((0,T), \mathbb{Y}\right)^2$ such that

$$\begin{aligned}
(C_\mathrm{m}\partial_t V, U)_{L^2(\Omega)} &= -\frac{1}{\chi}\left(\mathbf{G}_\mathrm{mono}\boldsymbol{\nabla}V, \boldsymbol{\nabla}U\right)_{L^2(\Omega)} - \left(I_\mathrm{ion}(V, \mathbf{s}) - I_\mathrm{app}, U\right)_{L^2(\Omega)} \ , \\
(\partial_t\mathbf{s}, U)_{L^2(\Omega)} &= (\mathbf{Z}(V, \mathbf{s}), U)_{L^2(\Omega)} \ ,
\end{aligned} \tag{2.19}$$

for all $U \in \mathbb{Y}$.

The quotient space $\mathbb{Y}/\mathbb{R} \subset H^1(\Omega)/\mathbb{R}$ is canonically isomorphic to the space of functions in $\mathbb{Y}$ with zero mean value.

As indicated above, the weak formulation of the bidomain and monodomain equations has two major drawbacks compared to the strong form. First, the evaluation of the non-linear term using summed quadrature

$$(I_\mathrm{ion}(V, \mathbf{s}), U)_{L^2(\Omega)} \approx \sum_i w_i \cdot I_\mathrm{ion}(V(\mathbf{x}_i), \mathbf{s}(\mathbf{x}_i))\, U(\mathbf{x}_i)$$

requires the evaluation of I_ion at multiple quadrature points per element, which is potentially costly. Second, since $\mathbf{Z}$ is a non-linear function, the term $(\mathbf{Z}(V, \mathbf{s}), U)_{L^2(\Omega)}$ cannot be expressed as a product of a mass matrix times a vector in such a way that the ordinary differential equations decouple after canceling the mass matrices on both sides of the equation.

A commonly used approximation (see, for example, Colli Franzone and Pavarino[46]) that addresses these two issues replaces the non-linear functions by appropriate approximations. More precisely, we replace

$$\begin{aligned} I_{\text{ion}}\left(\textstyle\sum_\alpha V_\alpha \pi_\alpha, \sum_\alpha \mathbf{s}_\alpha \pi_\alpha\right) &\longrightarrow \sum_\alpha I_{\text{ion}}(V_\alpha, \mathbf{s}_\alpha)\pi_\alpha \ , \\ \mathbf{Z}\left(\textstyle\sum_\alpha V_\alpha \pi_\alpha, \sum_\alpha \mathbf{s}_\alpha \pi_\alpha\right) &\longrightarrow \sum_\alpha \mathbf{Z}(V_\alpha, \mathbf{s}_\alpha)\pi_\alpha \ . \end{aligned} \tag{2.20}$$

If $\boldsymbol{\pi}$ is a nodal basis, the replacement functions are the nodal interpolations of the original functions. Inserting equation (2.20) into equations (2.17)–(2.19) we find that the non-linear terms require only $\dim(\mathbb{Y})$ evaluations of the functions I_{ion} and $\mathbf{Z}$, respectively. Similarly, the ordinary differential equations governing the state variables $\mathbf{s}$ decouple naturally.

With this approximation, and using the notations $\mathbf{M}$, $\mathbf{A}_{\text{i}}$, $\mathbf{A}_{\text{e}}$ and $\mathbf{A}_{\text{mono}}$ for the mass matrix, the discretized intra-cellular, extra-cellular and monodomain diffusion operator, respectively, we obtain the following equations in matrix-form.

Bidomain equation (parabolic-parabolic). Solve

$$C_{\text{m}} \begin{bmatrix} \mathbf{M} & -\mathbf{M} \\ -\mathbf{M} & \mathbf{M} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_{\text{i}} \\ \dot{\varphi}_{\text{e}} \end{bmatrix} = -\frac{1}{\chi} \begin{bmatrix} \mathbf{A}_{\text{i}} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{\text{e}} \end{bmatrix} \begin{bmatrix} \varphi_{\text{i}} \\ \varphi_{\text{e}} \end{bmatrix} - \begin{bmatrix} \mathbf{M} & -\mathbf{M} \\ -\mathbf{M} & \mathbf{M} \end{bmatrix} \begin{bmatrix} I_{\text{ion}} - I_{\text{app}} \\ \mathbf{0} \end{bmatrix} \tag{2.21}$$

coupled to a decoupled system of ordinary differential equations $\dot{\mathbf{s}}_\alpha = \mathbf{Z}\left((\varphi_{\text{i}})_\alpha - (\varphi_{\text{e}})_\alpha, \mathbf{s}_\alpha\right)$ (one for each basis function of $\mathbb{Y}$).

Bidomain equation (parabolic-elliptic). Solve

$$\begin{aligned} C_{\text{m}}\mathbf{M}\dot{V} &= \frac{-1}{\chi}\mathbf{A}_{\text{i}}\left(V + \varphi_{\text{e}}\right) - \mathbf{M}\left(I_{\text{ion}} - I_{\text{app}}\right) \ , \\ \left(\mathbf{A}_{\text{i}} + \mathbf{A}_{\text{e}}\right)\varphi_{\text{e}} &= -\mathbf{A}_{\text{i}} V \end{aligned} \tag{2.22}$$

coupled to ordinary differential equations $\dot{\mathbf{s}}_\alpha = \mathbf{Z}\left(V_\alpha, \mathbf{s}_\alpha\right)$.

Monodomain equation. Solve

$$C_{\text{m}}\mathbf{M}\dot{V} = -\frac{1}{\chi}\mathbf{A}_{\text{mono}} V - \mathbf{M}\left(I_{\text{ion}} - I_{\text{app}}\right) \tag{2.23}$$

coupled to ordinary differential equations $\dot{\mathbf{s}}_\alpha = \mathbf{Z}\left(V_\alpha, \mathbf{s}_\alpha\right)$.

Note that in equation (2.21) we padded the current vector on the right-hand side by $\mathbf{0}$ such that the mass matrices on the left and right side of the equation coincide. Using this trick only a single mass matrix needs to be assembled. The same idea is applied in the subsequent section in equations (2.28)– (2.31).

2.3.2 Temporal Discretization

In this section we discuss time discretization schemes that will be used in the later chapters. We present the discretization schemes within a finite element setting using the notation from the previous section. However, the same methods can be easily applied in the context of, e.g., a finite difference spatial discretization by replacing the mass matrix with the identity matrix.

The bidomain and monodomain equation can be solved using explicit, semi-implicit or implicit time discretization schemes. In general, low-order (first- or second-order) integration schemes appear to be the most popular choice in the literature. An exception is the use of higher-order Rosenbrock-type methods by Colli Franzone et al.[47]. Ethier and Bourgault[60] analyzed different time integration schemes for the bidomain equation (in parabolic-elliptic form) and found higher-order implicit-explicit methods to be the best choice when considering stability and accuracy criteria.

In this thesis we concentrate on first-order integration schemes based on explicit or implicit Euler schemes. Most of our results in Chapter 4 and Chapter 5 can be directly generalized to higher-order time discretization schemes.

An *explicit Euler* discretization for the bidomain equation in parabolic-elliptic form or the monodomain equation has been used, e.g., by Vigmond et al.[161] or Potse et al.[127]. The advantage of such a time discretization is its implementational simplicity and the possibility to achieve an overlap of communication and computation in parallel implementations[100,114]. However, due to the parabolic nature of the bidomain and monodomain equations, explicit discretization schemes are bound to the stability constraint

$$\tau \lesssim \delta^2 ,$$

where τ denotes the time step size and δ the minimal mesh width of the spatial discretization. This restriction renders explicit schemes inapplicable for studies relying on very high spatial resolution.

Explicit (and semi-implicit) low-order schemes are usually only applied for equations with decoupled state variable equations obtained by applying the "variational crime" (2.20). In this setting, the time discretization scheme is often combined with a first-order splitting between the parabolic-elliptic or parabolic equation and the governing equations for the state variables. Such a splitting allows for a flexible choice of the explicit integration scheme for the latter. In particular, the explicit Euler update for the gating variables can be replaced by a better alternative. One such option, that will be used throughout this thesis, is to employ *Rush-Larsen integration*[135] for gating variables. This integration scheme exploits the special form of the Hodgkin-Huxley equations to compute the updated gating variable by following the solution trajectory with a fixed membrane voltage. The Rush-Larsen update reads (cf. equation (2.6))

$$u^{i+1} = u_\infty(V^i) - \left(u_\infty(V^i) - u^i\right) \mathrm{e}^{-\tau/\tau_u(V^i)} . \tag{2.24}$$

Note that one recovers an explicit Euler update of u by using a two-term expansion of the exponential. The Rush-Larsen update is therefore more expensive than an explicit Euler update but enjoys better stability and accuracy properties. For time-dependent ionic concentrations, explicit integration schemes such as Runge-Kutta methods can be applied.

Below we state the formulas for an explicit Euler update from time step i to step $i+1$. Note that explicit integration is usually combined with mass lumping such that $\mathbf{M}$ is replaced by a diagonal matrix.

Bidomain equation (parabolic-elliptic). Update

$$C_\mathrm{m} V^{i+1} = C_\mathrm{m} V^i - \frac{\tau}{\chi} \mathbf{M}^{-1} \mathbf{A}_\mathrm{i} \left(V^i + \varphi_\mathrm{e}^i \right) - \tau \left(I_\mathrm{ion}^i - I_\mathrm{app}^i \right) \tag{2.25}$$

and subsequently solve

$$\left(\mathbf{A}_\mathrm{i} + \mathbf{A}_\mathrm{e} \right) \varphi_\mathrm{e}^{i+1} = -\mathbf{A}_\mathrm{i} V^{i+1} \,. \tag{2.26}$$

Update $\mathbf{s}_\alpha$ as in equation (2.24).

Monodomain equation. Update

$$C_\mathrm{m} V^{i+1} = C_\mathrm{m} V^i - \frac{\tau}{\chi} \mathbf{M}^{-1} \mathbf{A}_\mathrm{mono} V^i - \tau \left(I_\mathrm{ion}^i - I_\mathrm{app}^i \right) \,. \tag{2.27}$$

Update $\mathbf{s}_\alpha$ as in equation (2.24).

In contrast to explicit methods, the stability constraint of an implicit time discretization is independent of the spatial discretization. However, implicit models require the solution of high-dimensional, non-linear systems in each step. Fully implicit schemes have been used, for example, by Pavarino and Scacchi [121] and Colli Franzone et al. [47]. Note that in Ref. 47, the employed time-integration scheme allows for replacing the non-linear solver by a single Newton step. In order to lower the computational cost of a fully implicit scheme, Munteanu and Pavarino [116] proposed a decoupled scheme where only the membrane voltage is treated implicitly.

Below we state the formulas for an *implicit Euler* update from time step i to step $i+1$. To simplify the notation, we use matrix notation with the assumption that the non-linear terms have been approximated as specified in equation (2.20). This assumption, however, is made solely to simplify the notation.

Bidomain equation (parabolic-parabolic). Solve the non-linear system $\mathbf{F}\left(\varphi_\mathrm{i}^{i+1}, \varphi_\mathrm{e}^{i+1}, \mathbf{s}^{i+1} \right) = \mathbf{b}^i$ with

$$\begin{aligned} \mathbf{F}\left(\varphi_\mathrm{i}, \varphi_\mathrm{e}, \mathbf{s} \right) = &\left(\begin{bmatrix} C_\mathrm{m}\mathbf{M} & -C_\mathrm{m}\mathbf{M} & \mathbf{0} \\ -C_\mathrm{m}\mathbf{M} & C_\mathrm{m}\mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix} + \frac{\tau}{\chi} \begin{bmatrix} \mathbf{A}_\mathrm{i} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_\mathrm{e} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \right) \begin{bmatrix} \varphi_\mathrm{i} \\ \varphi_\mathrm{e} \\ \mathbf{s} \end{bmatrix} \\ &+ \tau \begin{bmatrix} \mathbf{M} & -\mathbf{M} & \mathbf{0} \\ -\mathbf{M} & \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{bmatrix} I_\mathrm{ion}\left(\varphi_\mathrm{i} - \varphi_\mathrm{e}, \mathbf{s} \right) \\ \mathbf{0} \\ -\mathbf{Z}\left(\varphi_\mathrm{i} - \varphi_\mathrm{e}, \mathbf{s} \right) \end{bmatrix} \,, \end{aligned} \tag{2.28}$$

$$\mathbf{b}^i = \begin{bmatrix} \mathbf{M} & -\mathbf{M} & \mathbf{0} \\ -\mathbf{M} & \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix} \left(\begin{bmatrix} C_\mathrm{m}\varphi_\mathrm{i}^i \\ C_\mathrm{m}\varphi_\mathrm{e}^i \\ \mathbf{s}^i \end{bmatrix} + \tau \begin{bmatrix} I_\mathrm{app}^{i+1} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \right) \,.$$

Bidomain equation (parabolic-elliptic). Solve the non-linear system $\mathbf{F}\left(V^{i+1}, \varphi_{\mathrm{e}}^{i+1}, \mathbf{s}^{i+1}\right) = \mathbf{b}^{i}$ with

$$\begin{aligned}\mathbf{F}\left(V, \varphi_{\mathrm{e}}, \mathbf{s}\right) = &\left(\begin{bmatrix} C_{\mathrm{m}}\mathbf{M} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix} + \frac{\tau}{\chi}\begin{bmatrix} \mathbf{A}_{\mathrm{i}} & \mathbf{A}_{\mathrm{i}} & \mathbf{0} \\ -\mathbf{A}_{\mathrm{i}} & (\mathbf{A}_{\mathrm{i}}+\mathbf{A}_{\mathrm{e}}) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}\right)\begin{bmatrix} V \\ \varphi_{\mathrm{e}} \\ \mathbf{s} \end{bmatrix} \\ &+ \tau\begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix}\begin{bmatrix} I_{\mathrm{ion}}\left(V, \mathbf{s}\right) \\ \mathbf{0} \\ -\mathbf{Z}\left(V, \mathbf{s}\right) \end{bmatrix}, \\ \mathbf{b}^{i} = &\begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix}\left(\begin{bmatrix} C_{\mathrm{m}}V^{i} \\ \mathbf{0} \\ \mathbf{s}^{i} \end{bmatrix} + \tau\begin{bmatrix} I_{\mathrm{app}}^{i+1} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}\right).\end{aligned} \tag{2.29}$$

Monodomain equation. Solve the non-linear system $\mathbf{F}\left(V^{i+1}, \mathbf{s}^{i+1}\right) = \mathbf{b}^{i}$ with

$$\begin{aligned}\mathbf{F}\left(V, \mathbf{s}\right) = &\left(\begin{bmatrix} C_{\mathrm{m}}\mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} \end{bmatrix} + \frac{\tau}{\chi}\begin{bmatrix} \mathbf{A}_{\mathrm{mono}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\right)\begin{bmatrix} V \\ \mathbf{s} \end{bmatrix} + \tau\begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} \end{bmatrix}\begin{bmatrix} I_{\mathrm{ion}}\left(V, \mathbf{s}\right) \\ -\mathbf{Z}\left(V, \mathbf{s}\right) \end{bmatrix}, \\ \mathbf{b}^{i} = &\begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} \end{bmatrix}\left(\begin{bmatrix} C_{\mathrm{m}}V^{i} \\ \mathbf{s}^{i} \end{bmatrix} + \tau\begin{bmatrix} I_{\mathrm{app}}^{i+1} \\ \mathbf{0} \end{bmatrix}\right).\end{aligned} \tag{2.30}$$

Semi-implicit integration schemes strive for combining the advantages of explicit schemes (simplicity and low cost per time step) with the advantages of implicit schemes (stability). In the following we will consider an *implicit-explicit (IMEX) Euler* scheme. For different semi-implicit discretizations we refer to Ethier and Bourgault[60]. In the implicit-explicit Euler discretization we present, the stiff diffusion operator is treated implicitly, while the non-linear current is treated explicitly. Moreover, the scheme is combined with a first-order splitting and Rush-Larsen integration for the gating variables. In case of the parabolic-elliptic formulation of the bidomain equation we also treat the extra-cellular potential explicitly in the parabolic equation. Therefore we need to solve two linear systems with block size one, instead of a single system with block size two.

Bidomain equation (parabolic-parabolic). Solve

$$\left(\begin{bmatrix} C_{\mathrm{m}}\mathbf{M} & -C_{\mathrm{m}}\mathbf{M} \\ -C_{\mathrm{m}}\mathbf{M} & C_{\mathrm{m}}\mathbf{M} \end{bmatrix} + \frac{\tau}{\chi}\begin{bmatrix} \mathbf{A}_{\mathrm{i}} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{\mathrm{e}} \end{bmatrix}\right)\begin{bmatrix} \varphi_{\mathrm{i}}^{i+1} \\ \varphi_{\mathrm{e}}^{i+1} \end{bmatrix} = \begin{bmatrix} \mathbf{M} & -\mathbf{M} \\ -\mathbf{M} & \mathbf{M} \end{bmatrix}\left(\begin{bmatrix} C_{\mathrm{m}}\varphi_{\mathrm{i}}^{i} \\ C_{\mathrm{m}}\varphi_{\mathrm{e}}^{i} \end{bmatrix} - \tau\begin{bmatrix} I_{\mathrm{ion}}^{i} - I_{\mathrm{app}}^{i} \\ \mathbf{0} \end{bmatrix}\right) \tag{2.31}$$

and update $\mathbf{s}_{\alpha}$ as in equation (2.24).

Bidomain equation (parabolic-elliptic). Solve

$$\left(C_{\mathrm{m}}\mathbf{M} + \frac{\tau}{\chi}\mathbf{A}_{\mathrm{i}}\right)V^{i+1} = C_{\mathrm{m}}\mathbf{M}V^{i} - \frac{\tau}{\chi}\mathbf{A}_{\mathrm{i}}\varphi_{\mathrm{e}}^{i} - \tau\mathbf{M}\left(I_{\mathrm{ion}}^{i} - I_{\mathrm{app}}^{i}\right) \tag{2.32}$$

and subsequently solve

$$\left(\mathbf{A}_{\mathrm{i}} + \mathbf{A}_{\mathrm{e}}\right)\varphi_{\mathrm{e}}^{i+1} = -\mathbf{A}_{\mathrm{i}}V^{i+1}. \tag{2.33}$$

Update $\mathbf{s}_{\alpha}$ as in equation (2.24).

Monodomain equation. Solve

$$\left(C_{\mathrm{m}}\mathbf{M}+\frac{\tau}{\chi}\mathbf{A}_{\mathrm{mono}}\right)V^{i+1}=C_{\mathrm{m}}\mathbf{M}V^{i}-\tau\mathbf{M}\left(I_{\mathrm{ion}}^{i}-I_{\mathrm{app}}^{i}\right)\ . \tag{2.34}$$

Update $\mathbf{s}_\alpha$ as in equation (2.24).

Note that the diffusion current is scaled by the inverse surface-to-volume ratio χ^{-1} in equations (2.31)–(2.34). Due to the size of χ and of the measured strength of the conductivity tensors in cardiac tissue, the mass-matrix terms are dominant for reasonable time step sizes τ. Therefore, the system matrices in equation (2.31), equation (2.32) and equation (2.34) are well conditioned and do not require complicated preconditioning techniques. Note, that this does not apply to equation (2.33). For this reason solving the parabolic-parabolic equation can be computationally less expensive even though the system matrix is larger[143].

2.4 Adaptive Computational Methods

In the previous section we have introduced numerical methods for the solution of the bidomain and monodomain equation. In this chapter we discuss the motivation for augmenting these techniques with adaptive control and review the current state of the research in this field.

2.4.1 Motivation

Adaptive solution techniques that adapt the computational mesh and/or the time step to the features of the solution are of interest for several reasons. They may allow for a more robust approximation of the considered phenomena, speed up the solution by reducing the required operations (e.g., by reducing the dimension of the ansatz/test spaces) or reduce the memory requirements, and hence allow for solving the same problem on smaller clusters of computers.

However, adaptive techniques incur an overhead due to the dynamic changes in the computational meshes and the need to iteratively improve meshes from an initial guess. Moreover, the need to use more complicated (unstructured) meshes or different discretization techniques can increase the memory requirements per degree of freedom compared to a uniform simulation. Hence, adaptive strategies can only be effective if the reduction in the degrees of freedom is sufficiently high. This, in turn, can only be the case if the (analytical) solution of the problem at hand exhibits localized features in space or time that need to be resolved by the numerical solution for an accurate approximation.

In Figure 2.3 we plot the solution of the monodomain equation at three different times. In this plot, each line corresponds to a level-set $\{V(\mathbf{x},t)=V_0\}$. Hence, regions of steep up-/or down-stroke are characterized by a high density of contour lines. From the visualization it is apparent that V features a high gradient in a relatively localized region (around the so-called *depolarization front*) but is smooth in the rest of the domain. Due to the (anisotropic) diffusion, the depolarization front moves through the domain. This "wave-like" shape of the solution motivates research into

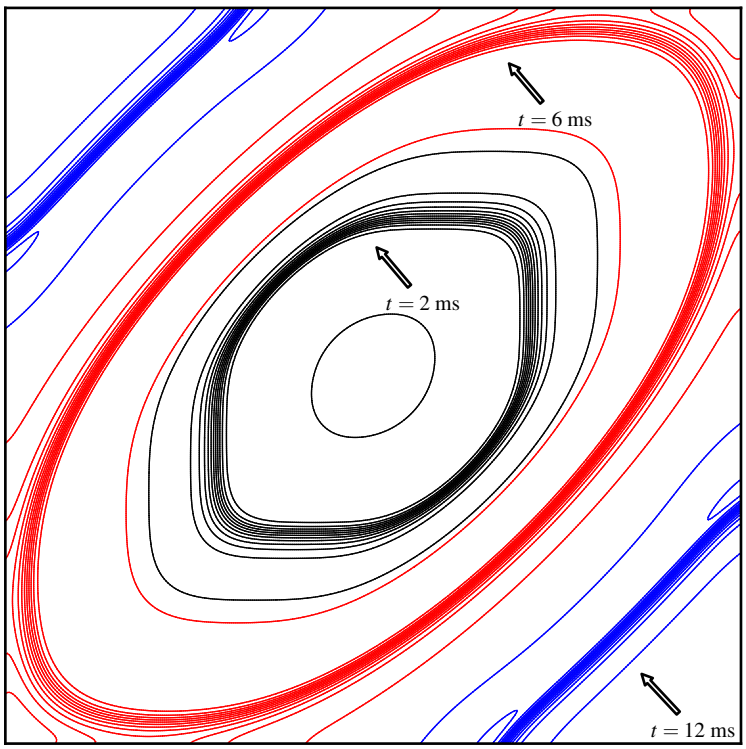


Figure 2.3. *Contour plot of the solution of the monodomain equation in a two-dimensional domain* $\Omega = (0,1)^2$ *at three different times. Lines represent the level-sets* $\{V(\mathbf{x},t) = V_0\}$ *for* V_0 *= -90 mV, -80 mV, . . . , 30 mV, 40 mV.*

spatial-adaptivity for the solution of the monodomain (and bidomain) equation. We will address this challenge in Chapter 4 and Chapter 5. Concerning temporal adaptivity, we already noted in Section 2.1.4 that the solutions of equation (2.3) exhibit fast changes during depolarization phase, followed by slow changes during the plateau and repolarization phase. Hence, adaptive time integration of (2.3) can be very effective since a much larger time step size τ can be used during the repolarization compared to the depolarization phase. However, when using a reaction-diffusion model with a spatial component, the depolarization front propagates through the domain such that global time step control is ineffective[166]. In Chapter 6 we address this problem by studying local time step control mechanisms.

2.4.2 Background

Over the past decades a vast number of adaptive discretization techniques have been developed so that an exhaustive discussion of the literature on this topic seems impossible. Instead, in the following we discuss several prominent methods that show different point of views on the problem of constructing adaptive methods and stand exemplarily for a subset of the research literature on adaptivity.

Block-structured *adaptive mesh refinement* (AMR) based on nested uniform meshes was first introduced by Brandt [33] in 1977. Berger and Oliger [22] and Berger and Colella [21] describe a practical algorithm for the solution of hyperbolic partial differential equations using block-structured adaptive meshes. The construction of finer meshes is based on the *clustering* of tagged elements into patches that are aligned with the elements of the coarser mesh. A finite volume discretization allows for an easy handling of the interface between coarse and fine meshes due to the flux-based formulation of the discrete equations. An estimate of the local truncation error, obtained by comparing the solutions on the fine and coarse mesh, is used to guide the adaptive refinement. Since 1989 block-structured adaptive mesh have been used in many publications (see Diachin et al. [56] and the references therein).

The parallelization and implementation of the Berger-Oliger-Collela method in high-quality software libraries has been discussed by several groups, see Refs. 45,81,131,169. Block-structured AMR algorithms have been shown to perform well on contemporary architectures and to be weakly scalable [159,169].

Due to the underlying assumption of a Cartesian grid, the handling of complicated geometries is not straightforward but possible, for example, by using an embedded boundary approach [142].

A related technique that also falls under the umbrella of structured adaptive mesh refinement techniques is *tree-based* AMR [139,157]. In contrast to the overlapping patch-based mesh handling, tree-based adaptive methods use binary space partitioning (BSP) trees to construct an adaptive mesh. Usually, quadtrees (in two dimensions) or octrees (in three dimensions) are used to organize the mesh. Tree-based AMR methods are mostly employed in the context of finite volume, discontinuous Galerkin or finite element methods with a one-to-one correspondence between tree leaves and mesh elements. Most publications use *balanced* trees that restrict the differences in levels of neighboring leaves to one, such that the differences in mesh width is at most a factor two.

Similar to block-structured AMR techniques, tree-based AMR methods cannot be directly applied to complicated geometries unless a suitable parametrization of the geometry over the unit cube is known. To address this problem, Burstedde et al. [35,36,38] developed algorithms for the management of *forests of octrees* built on conforming coarse tessellations of the computational domain via hexahedra. Recently this approach was implemented in the general purpose finite element code deal.II [13]. Tree-based structured AMR has been shown to scale well up to peta-scale class supercomputers [37,139].

Unstructured AMR algorithms take a different approach centered around the construction of conforming locally refined meshes and (usually nested) multi-level ansatz spaces. These methods often employ tetrahedral meshes and finite volume or finite element discretizations. Mesh refinement is steered by *a posteriori error estimators*, such as residual-based [59] or hierarchical [101] estimators. To obtain conforming finite elements in the presence of local refinement, special refinement rules are used to split neighbors of marked elements (*closures*). The finite element spaces built on refined meshes are naturally nested and can be used to implement multi-level solution methods.

Conforming unstructured AMR methods usually require complicated mesh management code (see, for example, Bastian et al.[14]) and tend to exhibit only low sustained performance and scalability on contemporary architectures.

Besides local mesh refinement, the approximation quality of the finite element spaces on unstructured meshes can also be adaptively controlled by varying the polynomial degrees (*p adaptivity*[7]) or by moving mesh nodes (*anisotropic adaptivity*[84]).

Recently, discontinuous Galerkin discretizations on non-conforming unstructured meshes have been investigated (see, for example, Gassner et al.[68]) as a way to achieve high performance and good scalability on unstructured adaptive meshes.

A different class of adaptive techniques uses *compression* algorithms to reduce the number of degrees of freedom required to achieve the desired accuracy. Harten[75] introduced a multiresolution algorithm based on a wavelet decomposition of the numerical solution. The solution is expanded into a linear combination of wavelet basis functions using the fast wavelet transformation and then truncated by dropping basis functions for which the coefficient is below a predefined tolerance. Since each basis function is associated with a grid node, this method results in an adaptive mesh that can be described via trees[44] or block-structured meshes[133]. Multiresolution analysis is usually used with finite volume or finite difference discretizations and explicit time integration schemes. A single integration step consists of a refinement step, where a grid is constructed that approximates the current and the next step with sufficient accuracy, the numerical integration and a compression step. Multiresolution analysis on block-structured meshes has been shown to perform well on multicore architectures and to be well suited for acceleration via graphics processing units[133,134]. As is the case for structured AMR methods, multiresolution analysis cannot be applied to complicated geometries straightforwardly.

A different approach, which is mostly used for high-dimensional problems, is based on *sparse grids*[178]. Sparse grids are constructed from tensor grids by dropping (nodal) basis functions with small support according to specific rules. One can show that for certain function classes, the truncation of the basis leads to only a small reduction in accuracy but a large reduction in the degrees of freedom. Ma and Zabaras[109] present an adaptive sparse-grid discretization for the solution of stochastic differential equations in the context of uncertainty quantification.

Adaptive mesh refinement techniques in cardiac simulation have been covered by a large number of publications. In the following we provide an overview about the literature.

Cherry et al.[40,41] use the Berger-Oliger-Collela AMR algorithm for solving the monodomain equation on two- and three-dimensional rectangular geometries. Speedups between 5 and 20 are reported for two-dimensional problems and a speedup of 50 is measured for a simple three-dimensional test problem.

Lines et al.[104] combine multi-level finite elements with wave-front tracking to solve the bidomain equation in parabolic-elliptic form. A semi-implicit time integration scheme is used and a multi-grid method is employed for solving the arising elliptic problems. For a two-dimensional test

problem, a speedup of about 2.2 is measured. This speedup appears to be caused by the reduction of the time required for the integration of the state variable ordinary differential equations in the phase-1 Luo-Rudy model which dominates the execution time.

Bendahmane et al.[20] use wavelet-based multiresolution analysis in combination with local time stepping to solve the monodomain and bidomain equations. The equations are discretized with a finite volume method and an explicit Euler or Runge-Kutta-Fehlberg time discretization for the parabolic problem. The elliptic problem in the parabolic-elliptic formulation of the bidomain equation is solved via Cholesky factorization. Speedups of 30 and 26 are reported for the solution of the two-dimensional monodomain and bidomain equation, respectively.

Whiteley[167] describes a two-level adaptive method for the solution of the bidomain equation in parabolic-elliptic form. Spatial refinement is controlled by the gradient of the extra-cellular potential or the membrane voltage. The high coarse-to-fine ratio is handled by imposing interpolated coarse values as Dirichlet boundary condition to the fine mesh.

Pennacchio[122,123] analyzes the mortar element method for the computation of extra-cellular potentials on statically refined non-conforming meshes.

Trangenstein and Kim[155] present a structured AMR algorithm for the solution of the monodomain equation with phase-1 Luo-Rudy membrane model. A second-order splitting is employed and the state variables are integrated with a singly diagonally implicit Runge-Kutta scheme. The parabolic equation is integrated with a Crank-Nicolson scheme and a multiplicative domain decomposition solver. The coarse-to-fine ratio is restricted to two and hanging nodes are handled via algebraic constraints. Local time stepping is used for the integration of the state variables. For a two-dimensional test problem, a speedup of up to 4.35 is reported.

Ying and Henriquez[175] describe an extension of this work to two- and three-dimensional body-fitted hexahedral finite elements. In this work the same second-order splitting is used but with an implicit integration scheme for the state variable ordinary differential equations that requires local Newton solves. The parabolic problem is solved at different time steps but always on the whole domain Ω. For a three-dimensional simulation of a dog ventricle, a speedup of 16.9 is reported.

Belhamadia[17] describes the use of anisotropic mesh adaptation for the bidomain equation with the FitzHugh-Nagumo membrane model (see Section 2.1.3). An implicit Euler time discretization is used. The mesh adaptation is driven by a hierarchical error estimator. The linear system is solved with a GMRES solver and ILU preconditioner. Belhamadia et al.[18] present results from three-dimensional simulations, including a realistic heart geometry. Speedup numbers are not mentioned for the heart geometry. For a simple three-dimensional geometry a speedup of 6.4 is reported.

Southern et al.[144,145] implemented anisotropic mesh adaptation in the CHASTE code and reported a speedup of 5–13 for the solution of the bidomain equation in parabolic-elliptic form on a realistic heart geometry using a Ruo-Ludy I membrane model. The authors also discuss the parallelization and report scaling results for the parallel code on up to 64 processes.

Colli Franzone et al.[47], Deuflhard et al.[55], Weiser et al.[166] study multi-level adaptive finite

elements for the solution of the monodomain and bidomain reaction-diffusion equations using the KARDOS code[101]. Low-order finite elements are used for the spatial discretization and a linear implicit time discretization of Rosenbrock type. In contrast to, e.g., an implicit Euler method these time discretization schemes require only the solution of a single linear system per time step. A hierarchical error estimator is used to guide the spatial adaptivity and global time step control is achieved via an embedded formula. Weiser et al.[166] report that for a fibrillation study, a reduction in the number of degrees of freedom by 150 is achieved but no gain in computing time was measured.

Despite the progress made in the field of adaptive discretizations for the mono- or bidomain equation, many open issues remain. Among the work listed above, only four groups (Refs. 18,145, 166,175) considered complicated three-dimensional geometries. Only Southern et al.[145] discuss the parallelization of their adaptive method. In many cases one notices a discrepancy between the numerical methods of choice in state-of-the-art computational models and the methods employed in the adaptive algorithms. Thus, it is unclear if the same reported speedup numbers hold in comparison to optimized uniform mesh solution methods.

In this thesis we compare the developed adaptive techniques to optimized uniform mesh solvers. In fact, in Chapter 4 and Chapter 5 we use a uniform mesh solver on a *structured* mesh for the comparison of execution times where possible. While structured mesh solvers are arguably less relevant for practical applications in computational electrocardiology, they provide an upper bound for the performance of computational heart models in practice. Throughout the text we promote the use of the execution time of a single or multiple time steps rather than end-to-end execution times for comparison of adaptive and uniform methods. Due to varying behavior of the solution process during the depolarization and repolarization phases we argue that accumulated execution times only provide an incomplete picture of the performance of an adaptive solution strategy.

We draw comparisons between our work and the work of others repeatedly in the discussion sections throughout the thesis.

3 Parallelization of the PROPAG Heart Model for Large-Scale Simulations

In this chapter, we discuss selected features of the PROPAG-5 cardiac simulation code. PROPAG[79,127,128] is a state-of-the-art computational heart model developed originally at the Université de Montréal. In the newest version of PROPAG several new features were introduced that allowed us to perform large scale simulations of unprecedented size. Here, we present and analyze two contributions by the author, namely the hybrid parallelization of the code and a parallel setup mechanism. Using PROPAG-5 we have been able to perform monodomain simulations with up to 1.5 billion mesh nodes, which is among the largest problem sizes reported in the literature for this scientific problem.

This chapter is an extended version of an article published in the proceedings of the second "Facing the multicore challenge" conference (see Ref. 100).

3.1 Characterization of PROPAG-4

The original code PROPAG-4 had been developed to solve both mono- and bidomain models on complicated geometries obtained from CT or MRI images of the heart. It was designed to run efficiently on shared-memory machines such as the SGI Altix family, using 16 to 128 cores. Parallelization had therefore been done with OpenMP directives in a NUMA-aware fashion (taking care of memory placement). In practice, PROPAG-4 can run heart models up to 100 million nodes in a reasonable amount of time and with good parallel performance. Strong scaling has a fixed limit of about $4 \cdot 10^5$ model nodes per core.

PROPAG works with semi-structured finite-difference meshes, i.e., many of the possible node positions are not occupied. The heart or torso anatomy is input as a Cartesian array storing the cell types (tissue type, blood, or void). We refer to the elements of this Cartesian box as *voxels* whereas non-void voxels are called *cells*. Based on the cell types of surrounding voxels, the vertices of the mesh receive types as well. Vertices that are not completely surrounded by void are referred to as (mesh) *nodes*. In PROPAG-4, connectivity is computed on the fly.

The code can run in three different modes. First, it can solve the monodomain equation (2.16) using an explicit Euler integration scheme (equation (2.27)). The control flow of the monodomain solver in PROPAG-4 is shown in Algorithm 3.1. The main loop is decomposed into so-called *laps*,

which are usually 10–50 time steps.

while $t < T$ **do**
  for $i = 1, \ldots, L_{\text{lap}}$ **do**
    Evaluate $I^i_{\text{dif}} = \chi^{-1} \nabla \cdot \mathbf{G}_{\text{mono}} \nabla V^i$
    Evaluate I^i_{app}
    Call `ion_step` to compute I^i_{ion} and to advance the state variables to the next time step using Rush-Larsen integration for the gating variables and an explicit Euler step for ionic concentrations
    Update $V^{i+1} = V^i + \tau \left(I^i_{\text{dif}} - I^i_{\text{ion}} + I^i_{\text{app}} \right)$
  end for
  Write the (downsampled) solution to disk
  $t \leftarrow t + \tau \cdot L_{\text{lap}}$
end while

Algorithm 3.1. *Monodomain solver in* PROPAG-*4.*

In bidomain mode, PROPAG-4 solves the bidomain equation (2.15) in parabolic elliptic form using operator splitting and an explicit Euler method for the parabolic equation. The control flow of the bidomain solver is shown in Algorithm 3.2.

while $t < T$ **do**
  for $i = 1, \ldots, L_{\text{lap}}$ **do**
    Evaluate $I^i_{\text{dif}} = \chi^{-1} \nabla \cdot \mathbf{G}_{\text{i}} \nabla \left(V^i + \varphi^i_{\text{e}} \right)$
    Evaluate I^i_{app} and call `ion_step`
    Update $V^{i+1} = V^i + \tau \left(I^i_{\text{dif}} - I^i_{\text{ion}} + I^i_{\text{app}} \right)$
    Solve $\nabla \cdot \left(\left(\mathbf{G}_{\text{i}} + \mathbf{G}_{\text{e}} \right) \nabla \varphi^{i+1}_{\text{e}} \right) = -\nabla \cdot \left(\mathbf{G}_{\text{i}} \nabla V^{i+1} \right)$
  end for
  Write the (downsampled) solution to disk
  $t \leftarrow t + \tau \cdot L_{\text{lap}}$
end while

Algorithm 3.2. *Bidomain solver in* PROPAG-*4.*

In the third (forward) mode, PROPAG-4 reads either the membrane voltage V or the current $I_{\text{rhs}} = -\nabla \cdot (\mathbf{G}_{\text{i}} \nabla V)$ and computes the extra-cellular potential from the equation

$$\nabla \cdot \left(\left(\mathbf{G}_{\text{i}} + \mathbf{G}_{\text{e}} \right) \nabla \varphi_{\text{e}} \right) = I_{\text{rhs}} \ .$$

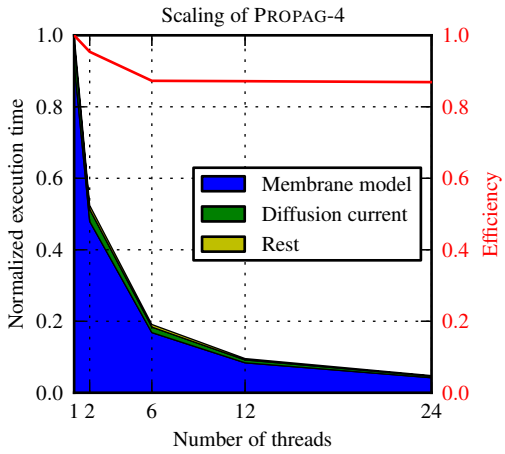


Figure 3.1. *Scaling of* PROPAG-*4 in a monodomain run with breakdown of runtime.*

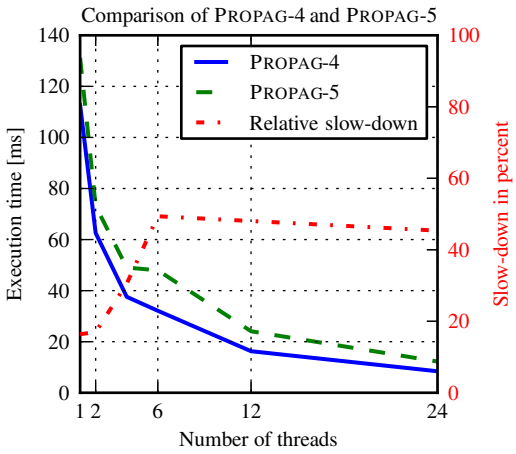


Figure 3.2. *Comparison of the timing for computing* I_{dif} *in* PROPAG-*4 and* PROPAG-*5.*

This mode is useful to compute the extra-cellular potential from a monodomain run. In many cases it is sufficient to compute φ_e on a coarser mesh than required for the propagation run.

PROPAG-4 implements multiple membrane models and allows for assigning different membrane models to different spatial regions. In this chapter all results will be presented using the ten Tusscher 2006 model, see Section 2.1.2. Note that PROPAG-4 uses tabulation to reduce the computational cost of the evaluation of the ionic current and the parameters in the Hodgkin-Huxley equations governing the evolution of the gating variables. In monodomain mode, the computation of I_{ion} and the state-variable update (both, in `ion_step`) dominate the runtime, see Figure 3.1.

The arising linear systems are solved using a Bi-CGSTAB solver (with restart capabilities) and an ILU(1) preconditioner. PROPAG-4 has been parallelized with OpenMP using mostly the `parallel for` worksharing construct. The ILU(1) preconditioner is implemented using a one-dimensional domain decomposition and parallel sections.

In Figure 3.1, an analysis of the runtime of PROPAG-4 is shown. The graph shows a breakdown of the runtime of a monodomain simulation on one 24-core node of a Cray XE6 (equipped with two AMD Opteron 2.1 GHz “Magny Cours” processors). Due to the NUMA-aware memory allocation and since runtime is distributed over only few scalable tasks of large granularity, the OpenMP parallelization is very efficient and OpenMP management overhead is negligible. The parallel efficiency on 24 cores is 86.9 % for this rather small example (422,091 mesh nodes).

3.2 Algorithms for Large-Scale Simulations

Solving very large problems introduces challenges that often require special care in the implementation or even changes in the (numerical) algorithms. In this section we describe two developments in PROPAG-5 that allowed us to perform monodomain production runs with systems in excess of one billion degrees of freedom.

3.2.1 Implicit-Explicit Euler Time Integration

As discussed in Section 2.3.2, explicit time integration schemes are limited in their applicability to high-resolution models due to the stability condition. Depending on the spatial resolution required for a specific simulation, implicit integration can thus be advantageous [132,168].

In PROPAG-5 we have implemented an implicit-explicit Euler time discretization which can be used as drop-in replacement for the explicit Euler integration scheme in PROPAG-4. We refer to Section 2.3.2 for a discussion of the implicit-explicit Euler discretization of the bidomain and monodomain equation. In our experience, the matrix in the arising linear system is well-conditioned for sufficiently small (but reasonable) time step sizes τ so that a few steps of the Bi-CGSTAB with Jacobi or Block-Jacobi ILU(0) preconditioner reduce the (relative) residual norm to the tolerance $\varepsilon = 10^{-8}$.

3.2.2 Parallel Setup

For an efficient end-to-end workflow, it is important to eliminate serial portions that can become bottlenecks during scale-out. A particular task that is frequently not parallelized in mesh-based codes is the initial I/O and mesh partitioning process. One reason for this is that in many cases it is hard to find a good initial distribution of the data. To address potential bottlenecks in the setup of PROPAG, we developed the following parallel "bootstrapping" algorithm.

The input to PROPAG are four files: One file stores voxel types and three files store the orientation of the tissue fibers $(\mathbf{a}_\mathrm{l}, \mathbf{a}_\mathrm{n}, \mathbf{a}_\mathrm{t})$ used to compute the conductivity tensor fields $\mathbf{G}_\mathrm{i}$, $\mathbf{G}_\mathrm{e}$ and $\mathbf{G}_\mathrm{mono}$. Each file describes a three-dimensional block of voxels with size $\mathtt{X} \times \mathtt{Y} \times \mathtt{Z}$. To partition this block before loading it, i.e., before knowing the actual distribution of the nodes, we use a three-dimensional Cartesian decomposition of the block. With calls to `MPI_File_set_view` and `MPI_Type_create_subarray` we change the views of the processes on the file so that the data can be read by means of a collective call to the function `MPI_File_read_all`. To simplify the bootstrapping procedure, each process reads an additional halo layer. Based on the voxel types, the list of mesh cells and mesh nodes are computed in parallel on each process. Since each process holds a Cartesian sub-tile of the $\mathtt{X} \times \mathtt{Y} \times \mathtt{Z}$-domain, it is trivial to compute connectivity information locally. Using these data we call PARMETIS to compute a new partition mapping `part` (or read a partition from a fifth input file).

Once the mapping of cells to the processes is known, the data is redistributed. The repartition algorithm in PROPAG-5 does not require any global numbering of the mesh entities but relies on a mapping from the current local indices to the local index on the target process. The Cartesian nature of the bootstrap decomposition provides the necessary communication mechanisms to exchange data between neighboring processes without the need to set up a convention for identifying local and remote mesh entities.

While it is possible to set up connectivity tables (node-to-node and node-to-cell) on the bootstrap mesh and exchange them during redistribution, we found this approach to be complex and error-prone. A much easier alternative with a negligible performance penalty is the use of an octree [39] to compute connectivity information after redistribution. This is possible due to the voxel

structure of the mesh. The octree in PROPAG-5 is built in integer-coordinate space and does not require floating-point operations.

The individual steps of the bootstrap and mesh distribution algorithm in PROPAG-5 are collected in Algorithm 3.3.

1: Read voxel-types using `MPI_File_read_all` and extract mesh entities
2: Compute connectivity between mesh cells and mesh nodes and compute `part` using `ParMETIS_V3_PartMeshKway`
3: Extend `part` to an array defined on all voxels in the local (to the process) sub-tile and exchange boundary values to fill in the halo
4: Compute number of peers and set up a mapping from cells to peers and from nodes to the list of peers
5: Identify nodes on inter-process boundaries and store their index after redistribution and the ranks of the processes storing a copy
6: Exchange data
7: Assign (consistently between processes) owners and mark inter-process connections as "in" and "out"
8: Build communication traces for in-going and out-going communications (Section 3.3)
9: Reorder the nodes in the communication trace according to coordinates to ensure consistency
10: Compute connectivity information using an octree
11: Reorder mesh entities locally (according to coordinates)

Algorithm 3.3. *Bootstrap and mesh distribution algorithm.*

Using this approach we have been able to bootstrap a mesh with 1.56 billion nodes ($\mathtt{X} = 2176, \mathtt{Y} = 1920, \mathtt{Z} = 3024$) in less than 79 seconds on 4224 cores of the Cray XT5 at CSCS (see Section 3.4). Roughly 19 seconds where required for Step 1 of the algorithm (corresponding to a read performance exceeding 0.6 GiB/s). The partition was read from a file (as PARMETIS was unable to partition such a large mesh, we computed `part` in a pre-processing step by interpolation from a coarser mesh) in about 47 seconds. The remaining portions of Algorithm 3.3 took about 12 seconds.

3.3 Hybrid Parallelization

The currently largest shared-memory machines are limited to a few thousand cores per machine while the largest distributed-memory architectures scale to millions of cores. To efficiently utilize these resources, we ported PROPAG-4 to an MPI code that can run on distributed-memory architectures. Such systems usually consist of a large number of multi-socket compute nodes connected by a high-speed interconnect. In recent years, the number of cores per socket has increased signifi-

cantly. Within a compute node, memory is shared between cores, usually with NUMA architecture. Therefore, we retained the existing OpenMP parallelization, which is efficient for intra-node parallelization, and added an MPI layer for inter-node parallelism. Such a *hybrid* parallelization approach has been used for a variety of codes and has proven beneficial for several reasons:

1. It simplifies adding new levels of concurrency beyond what is easily accomplished with MPI and hence can be used to overcome algorithmic scaling limitations (e.g., GTC[61]).

2. It allows to mitigate efficiency loss in applications that are limited by the scaling of all-to-all communication (e.g., PARATEC[119] and CPMD[88]) or where communication time is a significant part of the runtime.

3. Since the shared memory often renders halo (or overlap) zones unnecessary, hybrid codes can use less memory. If additional work must be performed on the halo, scalability can be enhanced by increasing the number of threads per process (e.g., FISH[94]).

4. It simplifies the load balancing of applications with dynamic or complicated structure since intra-process load balancing is possible using `dynamic` or `guided` loop scheduling (e.g., NPB BT-MZ Benchmark[130]).

It is worth noting, though, that hybrid parallelization is not always beneficial. Mahinthakumar and Saied[111] report no improvement in a hybrid implicit finite element solver. In general, there are many factors contributing to the performance of hybrid execution and results can vary between simulation setups[105].

3.3.1 MPI Parallelization

For the MPI parallelization of the code, we exploited techniques that have proven to be very efficient for the parallelization of general (unstructured) finite element applications. Hence, we use a cell-wise distribution of the geometry. The decomposition is computed through an interface to existing graph-partitioning libraries (e.g., PARMETIS[95]) as described in Section 3.2.2. Differently than previous versions of PROPAG, all arrays range only over cells and nodes and connectivity information is stored explicitly. Hence, the stencil-based computation of I_{dif} is replaced by a sparse-matrix vector multiplication. We use an ELLPACK-ITPACK format[136] that is suitable for vectorization by the compiler. In Figure 3.2 the impact of this change on the time required for computing I_{dif} is shown. The additional indirect addressing and the corresponding increase in memory bandwidth usage reduces performance which, however, is compensated for by better scalability of the MPI layer.

Since the mesh in PROPAG-5 is distributed cell-wise, nodes are duplicated on multiple processes. One of these processes is distinguished as the *owner* of the node. For inter-process communication, we use the notion of *communication traces* introduced by Sahni et al.[137]. In PROPAG-5 a communication trace consists of a set of nodes (located on an inter-process boundary) and the rank of a peer process. On the peer, a matching communication trace is built with a consistent ordering of the interface entities. Hence, by means of a communication trace, inter-process communication is

possible without the need for a global numbering of mesh entities. All communication is based on two primitives: The function `sumup_at_owner` gathers data on the owner and `copy_to_others` overwrites the data at each copy by the data at the owner (scatter). These communication steps are implemented on top of non-blocking MPI send/receive calls and an extended interface (`start`, `test`, `wait`) is provided to overlap these operations with computations.

Using these communication primitives, we can rewrite Algorithm 3.1 as shown in Algorithm 3.4. The algorithm is written in such a way that it allows for overlapping communication of the diffusion currents with the computation of I_{app} (to hide the communication in `sumup_at_owner`) and with the evaluation of I_{ion} for the interior nodes (to hide `copy_to_others`), assuming the necessary hardware capabilities. In our tests, we have not seen improvements in scalability or runtime due to overlap. Nevertheless, by construction, all receive calls are pre-posted timely before the `wait` call. This is important for good MPI performance on many systems including the targeted Cray XT5.

1: **while** $t < T$ **do**
2: **for** $i = 1, \ldots, L_{\text{lap}}$ **do**
3: Evaluate $I^i_{\text{dif}} = \chi^{-1}\nabla \cdot \mathbf{G}_{\text{mono}} \nabla V^i$ locally
4: Call `sumup_at_owner_start`(I^i_{dif})
5: Evaluate I^i_{app}
6: Call `sumup_at_owner_wait`(I^i_{dif})
7: Call `copy_to_others_start`(I^i_{dif})
8: Compute I^i_{ion} and integrate state variables for all owned nodes ▷ In `ion_step`
9: Call `copy_to_others_wait`(I^i_{dif}) ▷ In `ion_step`
10: Compute I^i_{ion} and integrate state variables for all other nodes ▷ In `ion_step`
11: Update $V^{i+1} = V^i + \tau \left(I^i_{\text{dif}} - I^i_{\text{ion}} + I^i_{\text{app}} \right)$
12: **end for**
13: Gather statistics using collective communication
14: Write the (downsampled) solution to disk
15: $t \leftarrow t + \tau \cdot L_{\text{lap}}$
16: **end while**

Algorithm 3.4. *Parallel monodomain solver in* PROPAG-*5.*

3.3.2 MPI Threading Support

The intra-process parallelization via OpenMP was retained and extended to new code segments. As in PROPAG-4, we mostly use `parallel for` worksharing constructs. This approach (in comparison to the use of large parallel sections) incurs some overhead but simplifies the implementation. Experiments with PROPAG-4 (Figure 3.1) show that OpenMP overhead does not significantly affect

the scalability of the explicit solver.

All MPI calls in PROPAG are performed outside the parallel sections. Therefore, the minimal level of thread support an MPI implementation must provide is `MPI_THREAD_FUNNELED`. As defined by the standard, this level of thread support suits applications where it is ensured that only the main thread makes MPI calls. In comparison to higher levels of thread support, this does not incur overhead due to locks/mutexes in the MPI implementation.

We do not anticipate savings in communication time by having multiple threads performing communication since the code is limited by latency rather than bandwidth. Using multiple threads for communication can be advantageous if a single thread is incapable of saturating the network interface [130].

3.4 Results

All experiments were performed on a Cray XT5 machine operated by the Swiss National Supercomputing Centre. The system consisted of 1844 nodes with two 6-core AMD Opteron 2.6 GHz "Istanbul" processors per node (22,128 cores in total). The nodes were connected through a Seastar 2+ interconnect. Due to an interconnect congestion problem at the time of the testing, we could not perform test on more than 8,448 cores.

For our experiments, we considered approximations of a model anatomy (based on CT data of a human heart obtained at autopsy [127]) at different spatial resolutions. We summarize the description of the four considered problem sizes (small, medium, large and extra-large) in Table 3.1.

Table 3.1. *Problem sizes for experiments.*

Name	Resolution	#cubes	#nodes
S	0.5 mm	3,024,641	3,200,579
M	0.25 mm	24,197,121	24,900,671
L	0.125 mm	193,576,968	196,390,842
XL	0.0625 mm	1,548,615,744	1,559,870,636

We studied strong scaling for the problem sizes **S**, **M**, **L** and **XL**, varying both the number of processes and the number of threads per process, the latter between 1 (one MPI process per core), 6 (one MPI process per socket), and 12 (one MPI process per node). For all setups we started with at least 12 threads. We measured the average time required to perform ten explicit Euler or implicit-explicit Euler steps, respectively. Every tenth step, an `MPI_Allreduce` was performed to sum up statistics that had been accumulated locally. For the purpose of our tests, we did not perform significant I/O. For the IMEX runs, we used the Bi-CGSTAB solver with a Jacobi preconditioner and a fixed time step size $\tau = 0.02$ ms.

Table 3.2. *Breakdown of communication time for **S** using explicit and implicit-explicit integration with one thread per process.*

#cores	% of walltime in point-to-point communication	% of walltime in collective communication
Explicit Euler		
132	4.91 %	2.31 %
1,056	20.10 %	10.07 %

#cores	% of walltime in point-to-point communication	% of walltime in collective communication
Implicit-Explicit Euler		
132	13.04 %	12.31 %
1,056	32.57 %	48.09 %

Table 3.3. *Characteristics of the node distribution during scale-out of **M**.*

#procs	12	24	528	1,056	4,224	8,448
% Increase in #nodes	1.58	2.33	10.14	13.25	22.55	29.67

3.4.1 Performance of Single-Threaded Execution

In Figure 3.3, the time per run for the different problem sizes is plotted against the number of threads (i.e., number of processes times threads per process). The code scales well up to 8,448 cores for the larger problem sizes. In general, the scaling of the explicit Euler is much better than the implicit-explicit Euler as the latter requires multiple `MPI_Allreduce` calls per time step and additional point-to-point communication for sparse matrix-vector multiplications.

For **S** on 1,056 cores (one thread per process), the implicit-explicit Euler requires about 169× more `MPI_Allreduce` calls than explicit Euler. At this scale, the code spends 48.0 % of the compute time in the calls to `MPI_Allreduce` (compared to 9.7 % for the explicit Euler). Hybrid execution can improve this situation, see Section 3.4.2. Nevertheless, for this small problem size, the code still achieves an efficiency of 56.5 % and 21.9 % on 1,056 cores using the explicit Euler and implicit-explicit Euler, respectively. For larger problems, such as **L**, the parallel efficiency on 8,448 cores relative to 132 cores (the minimum required to run the problem) is 81.6 % and 53.2 % for explicit Euler and implicit-explicit Euler, respectively.

The limits in (strong) scalability of PROPAG can be linked to two major sources of inefficiency: A relative increase in communication time and a sub-optimal decrease in the degrees of freedom per process.

In Table 3.2, we report the relative percentage of the average walltime of communication in the main computational loop as reported by the Integrated Performance Monitor (IPM)[90]. The data show that the relative communication time (both point-to-point and collective) increases by a factor of approximately 4 when increasing the number of cores by a factor of 8.

In Table 3.3 we show the increase in the total number of nodes due to the overlap between subdomains. Due to the cell-based decomposition, nodes on inter-process boundaries must be du-

Table 3.4. *Breakdown of communication time for **S** using explicit and implicit-explicit Euler. T_{Pt2Pt} and T_{Coll} denote point-to-point and collective communication time, respectively.*

#cores	procs × threads/proc	T_{Pt2Pt}	T_{Coll}
Explicit Euler			
132	132 × 1	5.12 s	2.41 s
	22 × 6	3.99 s	1.37 s
	11 × 12	4.87 s	2.34 s
1,056	1,056 × 1	3.90 s	1.95 s
	176 × 6	2.43 s	0.95 s
	88 × 12	2.25 s	0.76 s

#cores	procs × threads/proc	T_{Pt2Pt}	T_{Coll}
Implicit-Explicit Euler			
132	132 × 1	35.53 s	33.55 s
	22 × 6	20.86 s	25.89 s
	11 × 12	12.18 s	14.99 s
1,056	1,056 × 1	38.13 s	56.29 s
	176 × 6	13.73 s	39.81 s
	88 × 12	10.52 s	33.46 s

plicated so that the total number of nodes (where copies are accounted for) grows with the number of processes. As can be seen in Table 3.3, the number of nodes has grown by almost 30 % on 8,448 cores. Using an argument similar to that of Amdahl's law, we can derive an upper bound for the parallel efficiency as the ratio between the total number of nodes in serial and parallel. In our example, the maximum attainable efficiency when scaling from 12 to 8,448 cores is 78.3 %. A similar finding was reported by Sahni et al.[137] in the context of an unstructured finite element solver.

3.4.2 Benefits of Hybrid Execution

In the previous section, we have identified two major sources of scalability loss in PROPAG-5. In this section we will analyze how hybrid execution, using multiple threads per process, allows to mitigate these inefficiencies.

In Table 3.4, we present a breakdown of the communication time for the problem size **S**. The results for runs with one thread per process correspond to the results in Table 3.2. Unlike before, Table 3.4 contains absolute communication times (for 1010 time steps) to allow for comparing the results from different runs. Our results show that the use of multiple threads per process can significantly reduce the communication time. Using 6 or 12 threads per process reduces the time in `MPI_Allreduce` by 22–52 % or up to 61 %, respectively. Similarly, T_{Pt2Pt} is decreased by 22–64 % or 5–72 % for 6 or 12 threads. Interestingly though, a smaller number of processes does not always imply lower communication cost since the T_{Pt2Pt} for 11×12 threads is larger than for 22×6 threads. Using more threads per process leads to larger buffer sizes. This results in an improved bandwidth utilization but also increased latency.

In Section 3.4.1, we have noted that a strict upper limit for the parallel efficiency in PROPAG exists due to the growth of node copies on inter-process boundaries. For the intra-process parallelization based on OpenMP worksharing constructs, no overlap is required. When keeping the total number of threads constant, using more threads per process will result in fewer node copies. In Table 3.5, we show that this results in a strong reduction of the number of additional nodes. Consequently, the theoretical upper bound for the efficiency improves: When using 12 threads per

Table 3.5. *Percentage increase in #nodes for **M** with 1, 6, and 12 threads per process.*

#cores / threads	12	24	528	1,056	4,224	8,448
1	1.58	2.33	10.14	13.25	22.55	29.67
6	0.40	0.84	4.82	6.55	11.31	14.87
12	0.00	0.40	3.33	4.82	8.79	11.31

process, efficiency when going from 12 to 8,448 cores is bounded by 89.8 % (rather than 78.3 %, cf. Section 3.4.1). We measure an efficiency of 74 % for the explicit Euler solver which seems to be practically impossible to achieve with a pure MPI version.

The actual, measured improvement of the hybrid code (running with 6 or 12 threads per process, respectively) is shown in Figure 3.4. For problem size **M** and an explicit Euler discretization, threaded execution is beneficial starting at 96 cores. For the implicit Euler, which is more strongly limited by communication time, execution with 6 threads per process is advantageous already at 24 cores; execution with 12 threads per process is advantageous for 528 cores or more. When 2,112 cores or more are used, running with 12 threads per process is faster than running with 6 threads per process.

From Figure 3.5 the clear correlation between the efficiency of the pure MPI code and the improvement in performance due to hybrid execution is apparent. The correlation coefficient for this dataset is -0.915. On the other hand, hybrid execution (in particular when threading is used within a socket) can give a significant boost even in the range where the pure MPI code exhibits good parallel efficiency.

3.4.3 Weak Scaling of Monodomain Solver

In Table 3.6 we show weak scaling results for the implicit-explicit Euler discretization in PROPAG-5 when scaling from 132 cores with about 24 M nodes up to 8,448 cores and approximately 1.5 G nodes. These results show that preconditioning with a Block-Jacobi ILU(0) does not improve performance for modest system sizes but is crucial at large scale. For **XL**, the time per step is decreased by more than 40 % and the parallel efficiency is 56.4 % rather than 32.9 %.

The loss of efficiency can be partially explained by the increase in the number of iterations: On 132 cores, 28 time steps (of the first 90 steps) required a single Bi-CGSTAB iteration per step and 62 time steps required two iterations each. On 8,448 cores, 80 steps required three solver iterations each and the 10 remaining steps required four iterations per step. While it might be possible to improve the parallel efficiency of the time integration scheme by using more scalable preconditioners (e.g., domain decomposition preconditioners), it is unlikely that the actual runtime would be improved at this level of concurrency.

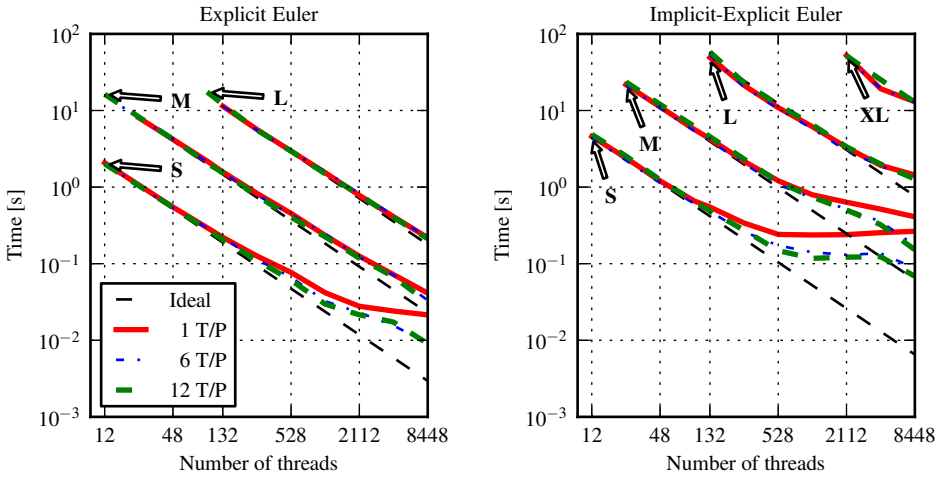


Figure 3.3. *Scaling of explicit Euler (left) and implicit-explicit Euler (right) on the Cray XT5. Problem **M** requires at least 24 cores for implicit-explicit or explicit Euler with one thread per process. **X** requires at least 132 cores for execution (96 when using 12 threads per process). The starting point for the strong scaling study for problem **XL** is 2112 cores.*

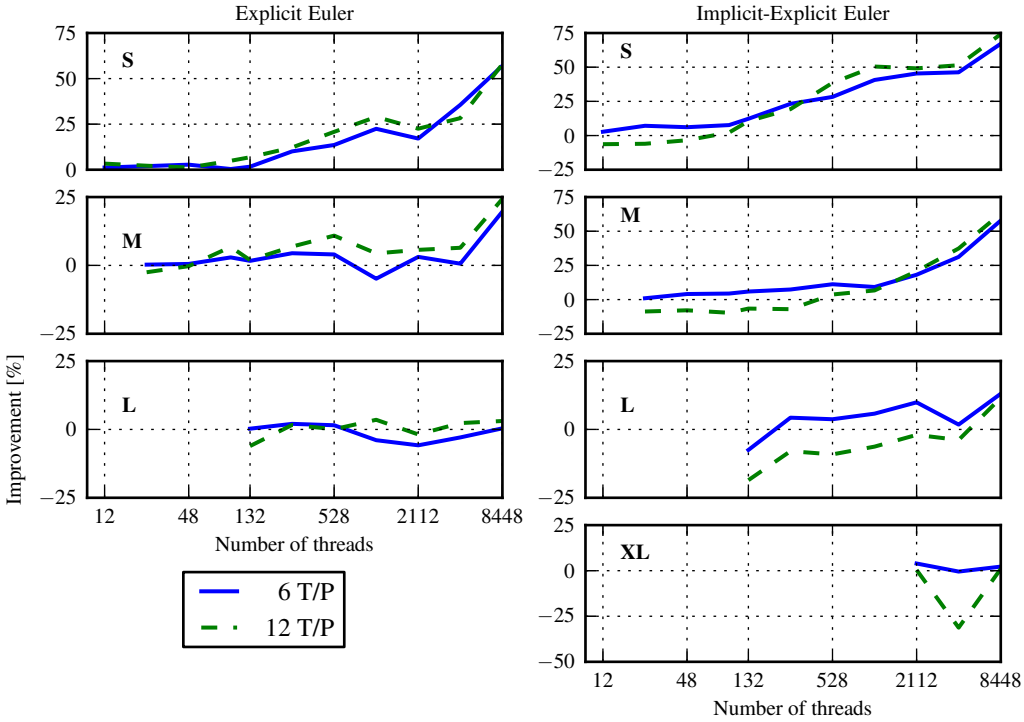


Figure 3.4. *Improvement through hybrid execution for explicit (left) and implicit-explicit Euler (right) relative to pure MPI for different problem sizes on the Cray XT5.*

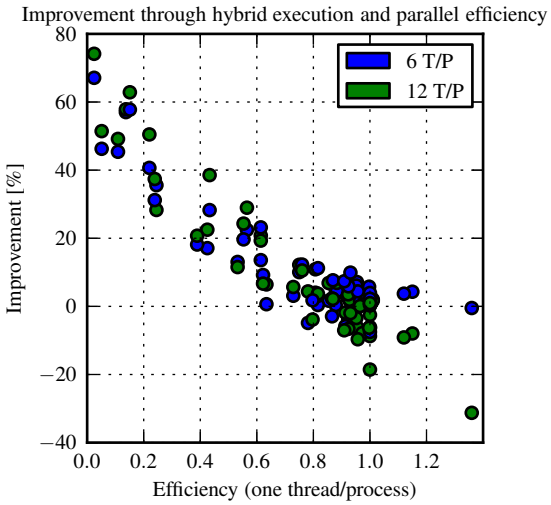


Figure 3.5. *Comparison of the improvement through hybrid execution and the efficiency of the pure MPI code. Data points are taken from both explicit and implicit-explicit Euler runs and include all four considered problem sizes.*

Table 3.6. *Weak scalability of the implicit-explicit Euler in* PROPAG-*5.*

#cores	problem size	Time per step w/o BJ ILU(0)	Time per step with BJ ILU(0)
132	**M**	0.43 s	0.44 s
1,056	**L**	0.61 s	0.49 s
8,448	**XL**	1.30 s	0.78 s

3.4.4 Performance of Parallel Setup

To assess the quality of the bootstrapping algorithm (Algorithm 3.3), we collected information about the bootstrap mesh for **M** on 12 to 8,448 cores on the Cray XT5.

Since the geometry data can be used for different simulation types (monodomain, bidomain or forward simulations), the simulation box sizes are not optimized for this monodomain run and only about 12 % of voxels in the simulation box are mesh cells. For example, the heart geometry contains atrial blood masses which are not taken into account in a monodomain simulation. Hence, our setup can be viewed as a worst-case scenario for the bootstrap algorithm; however, it is the most relevant case for production runs with PROPAG. Figure 3.6 depicts an indicator for the quality of the bootstrap distribution. We bin processes into four classes: *Idle* processes which own no cells; *Underutilized* processes which own less than $0.75\times$ the optimal value; *Overburdened* processes which own more than $1.25\times$ the optimal value and optimally loaded processes.

The quality of the bootstrap distribution deteriorates at increasing parallelism. As can be expected from geometric considerations, the number of idle threads grows quickly but reaches steady

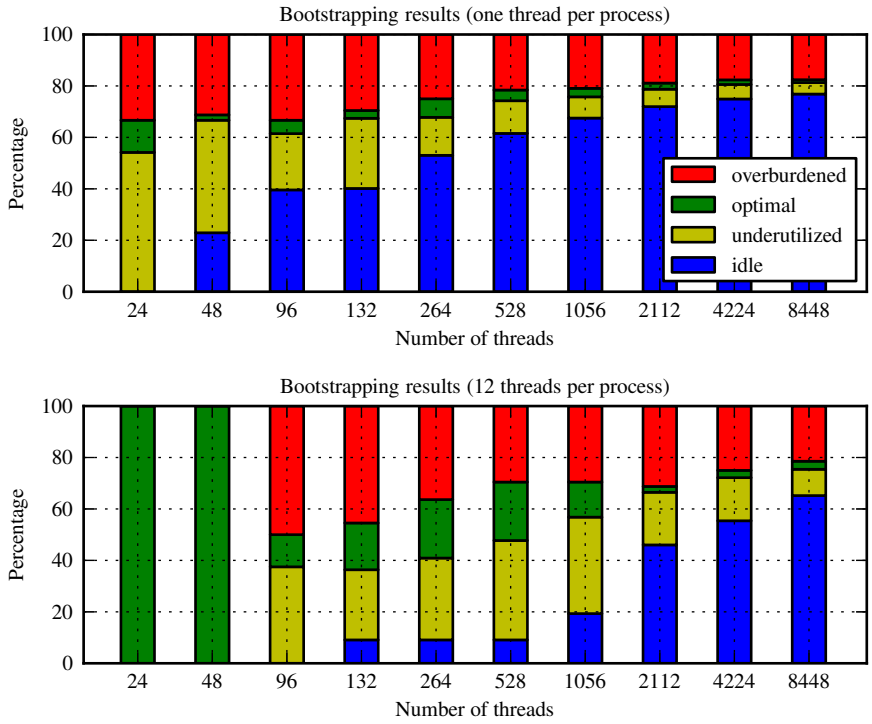


Figure 3.6. *Quality of the "best-effort" bootstrapping in* PROPAG*-5 when using one (top) and twelve (bottom) threads per process on the Cray XT5.*

state at 2,112 cores. Since the percentage of overburdened threads reaches a constant state, the memory imbalance is not prohibitive, i.e., PROPAG will still be able to bootstrap the mesh on higher core counts. When running with 8,448 processes, 76.8 % of processes are idle. In turn, this means that 1,956 processes are involved in the process: an improvement of more than two orders of magnitude compared to a serial alternative.

As can be seen from Figure 3.6, the use of more threads per process helps to mitigate the quality deterioration since the intra-process (thread level) distribution is not based on a Cartesian distribution but can be chosen optimally. However, because PARMETIS cannot use multiple threads per process, the hybrid code usually does not speed up the bootstrapping and mesh distribution phase in PROPAG-5.

3.5 Discussion

PROPAG-5 is a state-of-the-art computational heart model designed for efficient execution on current homogeneous architectures which nowadays feature intra-node parallelism through the use of multicore chips and inter-node parallelism through (high-speed) interconnection of multiple nodes. PROPAG-5 can efficiently utilize these systems by adapting the number of threads per process depending on the hardware characteristics.

Table 3.7. *Normalized throughput obtained from the lowest timing measured in Section 3.4.*

Problem size	Normalized throughput
Explicit Euler	
S	0.0217
M	0.0502
L	0.0594
XL	-

Problem Size	Normalized throughput
Implicit-Explicit Euler	
S	0.0029
M	0.0103
L	0.0100
XL	0.0079

While PROPAG-5 is competitive to other state-of-the-art implementations (see, for example, Refs. 115,118,125,160) it is unique in its ability to solve high-resolution models.

Recently, Mirin et al. [114] have demonstrated scalability of the CARDIOID model to 1 M cores on the BlueGene/Q *Sequoia* at Lawrence Livermore National Laboratory. This work exploits low-level optimizations targeted at the BG/Q architecture but also several optimizations that are implemented in PROPAG-5.

In this article, the authors publish a comparison of the throughput of several codes, including PROPAG-5. To compare different results from the literature, they used a *normalized throughput* defined as

$$\text{normalized throughput} = (60\,\text{s}) \cdot \left(\frac{500\,\text{ms}}{\tau} \cdot \frac{370 \cdot 10^6}{\#\,\text{cells}} \cdot (\text{time per step [s]}) \right)^{-1} . \tag{3.1}$$

When using the timings for problem size **XL** on 8,448 cores with the implicit-explicit Euler integrator, PROPAG-5 achieves a normalized throughput of only 0.0079 (which is close to the number used by Mirin et al. [114]). This number, however, is not representative for the performance of PROPAG-5 in production. On the one hand, our benchmark results where obtained with a time step size of 0.02 ms for both, the explicit and implicit-explicit Euler. For the latter, however, a larger time step size is advisable for production runs and would increase the normalized throughput. Moreover, our scaling runs where constrained to a maximum of 8,448 cores. For solving **XL** a larger number of cores would be employed when available. Our data suggest that PROPAG-5 could scale to at least 33,792 cores for this problem size. A more accurate representation of the performance of PROPAG-5 can obtained when using the other measurements from Section 3.4. In Table 3.7 we report the normalized throughput, as computed from equation (3.1) for explicit and implicit-explicit Euler for the different problem sizes. Note that Mirin et al. [114] report a normalized throughput of 0.018 for CARP [118]. In contrast to the data published by Niederer et al. [118], however, our timings do not include I/O overhead.

Besides the parallel bootstrapping algorithm and the hybrid parallelization discussed in this

chapter, PROPAG-5 moreover includes an optimized Lustre-aware I/O scheme and a new interface to PETSC[11] through which a variety of new preconditioners (e.g., the algebraic multi-grid BOOMERAMG[53,62]) can be used for solving the elliptic equation in the parabolic-elliptic formulation of the bidomain equation.

4 A Lightweight Adaptive Scheme for the Monodomain Equation

In this chapter we present a novel lightweight adaptive algorithm for solving the monodomain equation that was designed to exhibit good performance on contemporary homogeneous supercomputers. We present the results of several numerical experiments that allow to assess the performance of the proposed solution scheme.

4.1 Introduction

In Section 2.4.1 we motivated the study of adaptive techniques for the approximation of the monodomain equation. In the previous chapter we have presented performance and scalability results for a state-of-the-art uniform grid code. The fact that full heart simulations on uniform meshes require between $\mathcal{O}(10^6)$ and $\mathcal{O}(10^8)$ degrees of freedom and $\mathcal{O}(10^4)$ time steps clearly provides an incentive for studying adaptive strategies. At the same time, however, the good performance and excellent scalability of uniform grid codes present a challenge for adaptive methods that is not met by standard adaptive mesh refinement techniques [166].

We propose an adaptive scheme for time-dependent non-linear reaction-diffusion equations (with a focus on the monodomain equation) that was designed to exhibit a low memory footprint, to be well suited for contemporary central processing units and to be relatively simple to implement and parallelize. To this end we use non-conforming locally structured adaptive meshes and matrix-free block preconditioning. The use of a non-conforming discretization is central as it gives flexibility in the choice of the local mesh widths.

4.1.1 Overview

We consider the discretization of the monodomain equation (2.16) using finite elements and an implicit-explicit integration scheme as discussed in Section 2.3.2. Hence, the approximation V^{i+1} at the next time step is obtained as the solution of the variational problem: Find $V^{i+1} \in \mathbb{Y}$ so that

$$a\left(V^{i+1}, U\right) = b\left(U\right) \quad \text{for all } U \in \mathbb{Y}\,, \tag{4.1}$$

where $\mathbb{Y}$ denotes the spatial approximation space and

$$\begin{aligned} a(V,U) &= C_{\mathrm{m}}(V,U)_{L^2(\Omega)} + \frac{\tau}{\chi}\left(\mathbf{G}_{\mathrm{mono}}\boldsymbol{\nabla}V, \boldsymbol{\nabla}U\right)_{L^2(\Omega)} \\ b(U) &= C_{\mathrm{m}}\left(V^i, U\right)_{L^2(\Omega)} - \tau\left(I_{\mathrm{ion}}\left(V^i, \mathbf{s}^i\right) - I^i_{\mathrm{app}}, U\right)_{L^2(\Omega)} . \end{aligned} \tag{4.2}$$

The state variables $\mathbf{s} \in \mathcal{C}^1\left((0,T), \mathbb{Y}^S\right)$ are explicitly integrated.

We employ a *time window*-based dynamic adaptation procedure[36,148]. Instead of constructing new spatial approximation spaces in each time step we fix the approximation space over one so-called *lap*, which consists of $1 \leq L_{\mathrm{lap}} \in \mathbb{Z}$ time steps. The adaptation of the space is then based on accumulated error indicators. The integration over a lap is repeated multiple times to find an optimal approximation space that captures the solution behavior.

The advantage of this approach is that the overhead of the adaptive strategy (construction of a new approximation space and transfer of dynamic variables between spaces) is reduced. On the downside, however, this scheme typically leads to approximation spaces with higher dimensions.

In Algorithm 4.1 we present a schematic of the time integration scheme used in this chapter.

This chapter is organized as follows. In Section 4.2 we introduce the mesh data structure that is the foundation of the presented adaptive scheme. In the following Section 4.3 we discuss how to build (non-conforming) approximation spaces on these meshes. In Section 4.4 we present a tailored preconditioner for the system (4.1). Section 4.5 is concerned with transfer operators as used in Algorithm 4.1. Finally, in Section 4.6 we discuss error estimation and marking strategies.

4.2 Lightweight Adaptive Meshes

In classical *h*-adaptive unstructured (multi-level) finite element methods, the approximation space is usually chosen as piece-wise polynomials on a conforming mesh. Adaptation of the approximation space is achieved through local refinement and coarsening of the underlying mesh. This fine-grained control however comes at the expense of complex and inefficient data structures. Here, we follow an approach which trades control over the refinement process with the efficiency of the data structures. This is done by grouping mesh elements into batches that are collectively refined and coarsened. By using locally structured meshes and appropriate data structures we can efficiently handle these meshes.

In the following we assume that the computational domain Ω permits a conforming (in the sense of finite element meshes) tessellation

$$\overline{\Omega} = \bigcup_{i=1}^{N} \overline{\Omega_i} \tag{4.3}$$

into a finite number of *patches* Ω_i. Each patch shall be equivalent to $(0,1)^3$ up to translation and a trilinear mapping, i.e., there exists an invertible transformation from the unit cube to the patch, and

```
1: t ← 0
2: 𝕐 ← coarse approximations space
3: while t < T do
4:     Construct a smaller approximation space 𝕐′ by coarsening
5:     Transfer dynamic variables V, s to 𝕐′ (See Section 4.5) and update 𝕐 ← 𝕐′
6:     Save all dynamic variables
7:     for j = 1, ..., nrep + 1 do
8:         Clear all error indicator values.
9:         for i = 1, ..., L_lap do
10:            Evaluate right-hand side b as defined in equation (4.2)
11:            Integrate state variables s
12:            Solve for membrane voltage V (see Section 4.4)
13:            Accumulate error indicators (see Section 4.6.1)
14:        end for
15:        if j < nrep + 1 then
16:            Construct a larger approximation space 𝕐′ by refinement (see Section 4.6.2)
17:            break if 𝕐′ equals 𝕐 or if the estimated error is small enough
18:            Restore the saved variables V, s and transfer them to 𝕐′. Update 𝕐 ← 𝕐′
19:        end if
20:    end for
21:    t ← t + L_lap · τ
22: end while
```

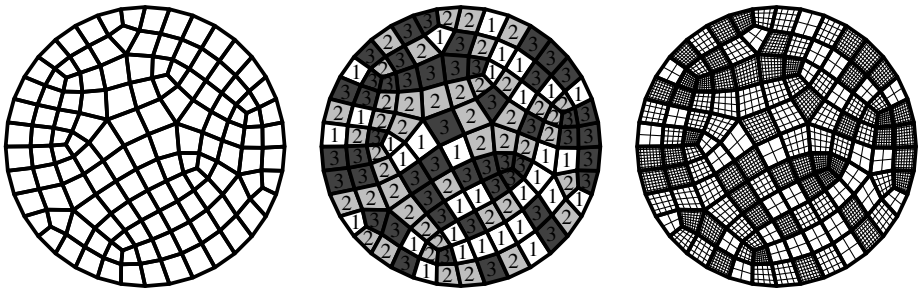
Algorithm 4.1. *Time integration algorithm (schematic).*

hence Ω_i has the shape of a hexahedron. We moreover assume that this transformation has positive determinant.

In our adaptive scheme, each patch inherits a structured mesh from the unit cube via the transformation $(0,1)^3 \to \Omega_i$. However, each patch is individually meshed with no regards to conformity on the interface between adjacent patches. More precisely, let $(\delta_\ell)_{\ell\geq 1} \subset (0, \frac{1}{2}]$, be an arbitrary (but fixed) sequence of admissible mesh widths. Each δ_ℓ defines a structured mesh $\hat{\mathscr{T}}_{\delta_\ell}$ on $(0,1)^3$ with edge length δ_ℓ. In our adaptive algorithm we choose a *level* $1 \leq \ell_i \in \mathbb{Z}, 1 \leq i \leq N$, individually on each patch, allowing us to resolve spatial features of the PDE solution at the optimal resolution. We obtain a tensor-structured mesh $\mathscr{T}_{\ell_i}$ on Ω_i as the image of $\hat{\mathscr{T}}_{\delta_{\ell_i}}$ under the transformation from $(0,1)^3$ to Ω_i. In the following we always assume $\delta_\ell = 2^{-j}$ for some $1 \leq j \in \mathbb{Z}$ but our approach could be generalized to other choices.

In this setting, the *mesh* $\mathscr{T}_{\boldsymbol{\ell}}$ is uniquely defined by the vector $\boldsymbol{\ell} = (\ell_i)_{i=1}^N$ of patch levels or equivalently by the local mesh widths $\boldsymbol{\delta} = (\delta_{\ell_i})_{i=1}^N$. In a time-dependent simulation, $\boldsymbol{\ell}$ and $\boldsymbol{\delta}$ are piece wise constant time-dependent functions.

Figure 4.1 shows a sketch of the mesh creation process starting from the coarse tessellation, the assignment of a level to each patch and the mapping of structured meshes to each patch.



(a) *Decomposition of Ω into patches Ω_i.*

(b) *Assignment of levels $1 \leq \ell_i \leq 3$ to patches.*

(c) *Patch-wise structured non-conforming mesh.*

Figure 4.1. *Two-dimensional sketch of the geometric setup.*

4.3 Mortar Discretization

Associated with the mesh $\mathscr{T}_{\ell_i}$ on Ω_i we define a local approximation space

$$\mathbb{X}_{\ell_i} = \left\{ u \in \mathcal{C}^0(\Omega_i) \mid u|_E \in \mathbb{Q}_1(E) \text{ for all } E \in \mathscr{T}_{\ell_i} \right\},$$

where $\mathbb{Q}_1(E)$ denotes the space of functions on the element E whose pull-backs are trilinear polynomials on the reference element[59].

Using the spaces $\mathbb{X}_{\ell_i}$ we form the product space

$$\mathbb{X}_{\boldsymbol{\ell}} = \prod_{i=1}^{N} \mathbb{X}_{\ell_i} .$$

We modify the definition of the bilinear form a by replacing the L^2-scalar product with a broken L^2-product for the second-order term, i.e.,

$$a(V,U) = \sum_{i=1}^{N} C_{\mathrm{m}}(V,U)_{L^2(\Omega_i)} + \frac{\tau}{\chi} \sum_{i=1}^{N} (\mathbf{G}_{\mathrm{mono}} \boldsymbol{\nabla} V, \boldsymbol{\nabla} U)_{L^2(\Omega_i)} \tag{4.4}$$

The definition of a in equation (4.4) and equation (4.2) are equivalent for functions in $H^1(\Omega)$ but the former is also well defined for the non-conforming space $\mathbb{X}_{\boldsymbol{\ell}} \not\subset H^1(\Omega)$.

It is well known that $\mathbb{X}_{\boldsymbol{\ell}}$ is not well suited for the discretization of equation (4.1) since it provides no control over the jump of functions on the interface between adjacent patches and hence one cannot bound the consistency error in terms of the mesh size[172].

More precisely, according to Strang's second lemma

$$\|V^{i+1} - V_{\boldsymbol{\ell}}^{i+1}\| \lesssim \inf_{U_{\boldsymbol{\ell}} \in \mathbb{X}_{\boldsymbol{\ell}}} \|V^{i+1} - U_{\boldsymbol{\ell}}\| + \sup_{U_{\boldsymbol{\ell}} \in \mathbb{X}_{\boldsymbol{\ell}}} \frac{|b(U_{\boldsymbol{\ell}}) - a(V^{i+1}, U_{\boldsymbol{\ell}})|}{\|U_{\boldsymbol{\ell}}\|} . \tag{4.5}$$

Since, $\mathbb{X}_\ell \not\subset H^1(\Omega)$ the second term (the *consistency error*) does not vanish, even with exact evaluation of the bilinear form a. The solution $V^{i+1} \in H^1(\Omega)$ of equation (4.1) satisfies

$$\sum_{i=1}^{N} \left[C_{\mathrm{m}} \left(V^{i+1}, U_\ell \right)_{L^2(\Omega_i)} + \frac{\tau}{\chi} \left(\mathbf{G}_{\mathrm{mono}} \boldsymbol{\nabla} V^{i+1}, \boldsymbol{\nabla} U_\ell \right)_{L^2(\Omega_i)} \right] + \int_{\mathcal{S}} (\mathbf{n} \cdot \mathbf{G}_{\mathrm{mono}} \boldsymbol{\nabla}) V^{i+1} \, [U_\ell] \, \mathrm{d}S(\mathbf{x}) = b(U_\ell) \ , \tag{4.6}$$

where $[\cdot]$ denotes the jump of a function across an interface, Γ_{ij} denotes the interior of $\partial\Omega_i \cap \partial\Omega_j$ and

$$\overline{\mathcal{S}} = \bigcup_{i,j=1}^{N} \overline{\Gamma_{ij}} \tag{4.7}$$

denotes the *skeleton* of codimension one. Inserting (4.6) into (4.5) we obtain

$$\sup_{U_\ell \in \mathbb{X}_\ell} \frac{|b(U_\ell) - a(V^{i+1}, U_\ell)|}{\|U_\ell\|} = \sup_{U_\ell \in \mathbb{X}_\ell} \frac{\int_{\mathcal{S}} (\mathbf{n} \cdot \mathbf{G}_{\mathrm{mono}} \boldsymbol{\nabla}) V^{i+1} \, [U_\ell] \, \mathrm{d}S(\mathbf{x})}{\|U_\ell\|} \ . \tag{4.8}$$

This term cannot be bounded as a (non-trivial) function of the mesh size $\boldsymbol{\delta}$.

The *mortar finite element* method[25] introduces constraints that define a subspace of $\mathbb{X}_\ell$ suitable for the discretization of elliptic problems, even in the presence of large variations in the mesh width. These constraints state that the jump of trial functions, tested against functions in a multiplier space, must vanish. In view of equation (4.8) this multiplier space must be chosen so that it can approximate the trace of $(\mathbf{n} \cdot \mathbf{G}_{\mathrm{mono}} \boldsymbol{\nabla}) V$ on the skeleton. In the following we briefly review the key aspects of the mortar element method as we employ it. For more information we refer to Bernardi et al.[26].

4.3.1 Mortar Constraints

On the interface Γ_{ij}, two potentially different hyper-surface meshes are induced by the adjacent patches. For each $\Gamma_{ij} = \Gamma_{ji}$ we designate one side as *mortar* (or *master*) side whereas the other side is termed *non-mortar* (or *slave*). This choice induces non-overlapping decompositions

$$\overline{\mathcal{S}} = \bigcup_{m=1}^{M} \overline{\gamma_m^-} \quad \text{and} \quad \overline{\mathcal{S}} = \bigcup_{m=1}^{M} \overline{\gamma_m^+}$$

of the skeleton into the mortars and non-mortars, respectively. Here, $\gamma_m^- = \Gamma_{ij}$ and $\gamma_m^+ = \Gamma_{ji}$ or vice versa. In our method, the mortar side is always associated with the finer mesh, i.e., $\gamma_m^- = \Gamma_{ij}$ if $\delta_{\ell_i} < \delta_{\ell_j}$. If $\delta_{\ell_i} = \delta_{\ell_j}$ we make an arbitrary (but fixed) choice.

The following sets of mesh nodes are used subsequently. Note that throughout this chapter we use small Greek letters for mesh nodes and use a dot to identify nodes on slave sides.

$$\begin{aligned}
\mathcal{N}_{\ell_i} &= \text{Mesh nodes of } \mathcal{T}_{\ell_i} \,, \\
\mathcal{N}_{\ell_i}^{\circ} &= \text{Interior mesh nodes of } \mathcal{T}_{\ell_i} \,, \\
\mathcal{N}_{\gamma_m^-} &= \text{Mesh nodes on } \gamma_m^- \text{ (induced by master side)} \,, \\
\mathcal{N}_{\gamma_m^+} &= \text{Mesh nodes on } \gamma_m^+ \text{ (induced by slave side)} \,, \\
\mathcal{N}_{\gamma_m^+}^{\circ} &= \{ \dot{\alpha} \in \mathcal{N}_{\gamma_m^+} \mid \dot{\alpha} \in \gamma_m^+ \setminus \partial \gamma_m^+ \} \,, \\
\mathcal{N}_{\gamma_m^+}^{\partial} &= \mathcal{N}_{\gamma_m^+} \setminus \mathcal{N}_{\gamma_m^+}^{\circ} \,.
\end{aligned}$$

By $\boldsymbol{\theta} = (\theta_\alpha)_{\alpha \in \bigcup_{i=1}^{N} \mathcal{N}_{\ell_i}}$ we denote the nodal basis of $\mathbb{X}_{\boldsymbol{\ell}}$.

To define a suitable subspace $\mathbb{Y}_{\boldsymbol{\ell}}^{\mathrm{m}} \subset \mathbb{X}_{\boldsymbol{\ell}}$, we choose a discrete *Lagrange multiplier space* $\mathbb{M}_{\gamma_m^+} \subset L^2(\gamma_m^+)$ for each non-mortar γ_m^+. We define

$$\mathbb{M}_{\gamma_m^+} = \operatorname{span} \{ \psi_{\dot{\alpha}} \}_{\dot{\alpha} \in \mathcal{N}_{\gamma_m^+}^{\circ}} \,,$$

i.e., we associate one basis function with each mesh node located in the interior of the non-mortar side. As we retain the degrees of freedom associated with mesh nodes on the *wire basket* $\bigcup_{m=1}^{M} \partial \gamma_m^+$, the space $\mathbb{M}_{\gamma_m^+}$ has the correct dimension. To compensate for the fact that no multipliers are associated with the nodes on $\partial \gamma_m^+$, we need to modify the basis functions in boundary elements in order to preserve the approximation properties of the multiplier space. The choice of the basis functions $\{\psi_{\dot{\alpha}}\}$ is discussed in Section 4.3.3.

We call a function $U_{\boldsymbol{\ell}} \in \mathbb{X}_{\boldsymbol{\ell}}$ *admissible* if

$$\int_{\gamma_m^+} [U_{\boldsymbol{\ell}}] \cdot \mu \, \mathrm{d}S(\mathbf{x}) = 0 \quad \text{for all } \mu \in \mathbb{M}_{\gamma_m^+} \text{ and } m = 1, \ldots, M \,. \tag{4.9}$$

For the discretization of (4.1) we use the following *constrained space* (space of admissible functions) as ansatz and test space:

$$\mathbb{Y}_{\boldsymbol{\ell}}^{\mathrm{m}} = \{ U_{\boldsymbol{\ell}} \in \mathbb{X}_{\boldsymbol{\ell}} \mid \text{Equation (4.9) holds for } U_{\boldsymbol{\ell}} \} \,.$$

For our choice of $\mathbb{M}_{\gamma_m^+}$ we have $\mathbb{Y}_{\boldsymbol{\ell}}^{\mathrm{m}} \subset \mathcal{C}^0(\Omega)$ only in case of $\ell_i = \ell_j$ for all i, j.

4.3.2 Mortar Projection

Let us briefly discuss the algebraic representation of the constraints (4.9). For a mortar $\gamma_m^- = \Gamma_{ij}$ we can write the values of $U_{\boldsymbol{\ell}} \in \mathbb{X}_{\boldsymbol{\ell}}$ on the mortar side as $U_{\boldsymbol{\ell}}|_{\gamma_m^-} = \sum_\alpha (U_{\boldsymbol{\ell}})_\alpha^{\mathrm{m}} \theta_\alpha$. Similar, on the non-mortar side, we can write $U_{\boldsymbol{\ell}}|_{\gamma_m^+} = \sum_{\dot{\alpha}} (U_{\boldsymbol{\ell}})_{\dot{\alpha}}^{\mathrm{nm}} \theta_{\dot{\alpha}}$. Inserting $\mu = \psi_{\dot{\beta}}$ in equation (4.9) we obtain

$$\begin{aligned}
0 &= \int_{\gamma_m^+} \Big(\sum_{\dot{\alpha}} (U_{\boldsymbol{\ell}})_{\dot{\alpha}}^{\mathrm{nm}} \theta_{\dot{\alpha}} - \sum_{\alpha} (U_{\boldsymbol{\ell}})_\alpha^{\mathrm{m}} \theta_\alpha \Big) \psi_{\dot{\beta}} \, \mathrm{d}S(\mathbf{x}) \\
&= \int_{\gamma_m^+} \sum_{\dot{\alpha}} (U_{\boldsymbol{\ell}})_{\dot{\alpha}}^{\mathrm{nm}} \theta_{\dot{\alpha}} \psi_{\dot{\beta}} \, \mathrm{d}S(\mathbf{x}) + \int_{\gamma_m^+} \Big(\sum_{\dot{\alpha} \in \mathcal{N}_{\gamma_m^+}^{\partial}} (U_{\boldsymbol{\ell}})_{\dot{\alpha}}^{\mathrm{nm}} \theta_{\dot{\alpha}} - \sum_{\alpha} (U_{\boldsymbol{\ell}})_\alpha^{\mathrm{m}} \theta_\alpha \Big) \psi_{\dot{\beta}} \, \mathrm{d}S(\mathbf{x}) \,.
\end{aligned}$$

We introduce the matrices $\mathbf{D}$, $\mathbf{R}$, $\mathbf{C}$ with

$$\mathbf{D}_{\dot\alpha\dot\beta} = \int_{\gamma_m^+} \psi_{\dot\alpha}\theta_{\dot\beta}\,\mathrm{d}S(\mathbf{x})\,, \tag{4.10a}$$

$$\mathbf{R}_{\dot\alpha\varepsilon} = \int_{\gamma_m^+} \psi_{\dot\alpha}\theta_{\varepsilon}\,\mathrm{d}S(\mathbf{x})\,, \tag{4.10b}$$

$$\mathbf{C}_{\dot\alpha\dot\nu} = -\int_{\gamma_m^+} \psi_{\dot\alpha}\theta_{\dot\nu}\,\mathrm{d}S(\mathbf{x})\,, \tag{4.10c}$$

for $\dot\alpha,\dot\beta \in \mathcal{N}^\circ_{\gamma_m^+}$, $\varepsilon \in \mathcal{N}_{\gamma_m^-}$ and $\dot\nu \in \mathcal{N}^\partial_{\gamma_m^+}$. The *mortar projection* is represented by $\mathbf{P} = \mathbf{D}^{-1}\begin{bmatrix}\mathbf{R} & \mathbf{C}\end{bmatrix}$. Introducing $\mathbf{U}^{\mathrm{S}} = ((U_\ell)^{\mathrm{nm}}_{\dot\alpha})_{\dot\alpha\in\mathcal{N}^\circ_{\gamma_m^+}}$ and

$$\mathbf{U}^{\mathrm{M}} = \begin{bmatrix} ((U_\ell)^{\mathrm{m}}_{\varepsilon})_{\varepsilon\in\mathcal{N}_{\gamma_m^-}} \\ ((U_\ell)^{\mathrm{nm}}_{\dot\nu})_{\dot\nu\in\mathcal{N}^\partial_{\gamma_m^+}} \end{bmatrix}\,,$$

we can rewrite equation (4.9) as $\mathbf{U}^{\mathrm{S}} = \mathbf{P}\mathbf{U}^{\mathrm{M}}$.

4.3.3 Dual Lagrange Multipliers

The first multiplier spaces used in the mortar element method[25] were standard nodal functions (with modifications at the boundary). The disadvantage of this choice for $\mathbb{M}_{\gamma_m^+}$ is that $\mathbf{D}^{-1}$ is a full matrix for three-dimensional problems and hence the mortar projection is full as well.

In this work, we use *dual Lagrange multipliers*[171,172] to span the multiplier space $\mathbb{M}_{\gamma_m^+}$ which have the advantage that the mortar projection is sparse and (equivalently) that the resulting nodal basis functions of $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}$ have local support. Dual multipliers are characterized by the *biorthogonality condition*

$$\int_{\gamma_m^+} \psi_{\dot\alpha}\theta_{\dot\beta}\,\mathrm{d}S(\mathbf{x}) = \delta_{\dot\alpha\dot\beta}\int_{\gamma_m^+}\theta_{\dot\beta}\,\mathrm{d}S(\mathbf{x}) \quad \text{for } \dot\alpha,\dot\beta\in\mathcal{N}^\circ_{\gamma_m^+}\,. \tag{4.11}$$

Biorthogonality does not hold for $\dot\beta \in \partial\gamma_m^+$. Inserting equation (4.11) into the definition of $\mathbf{D}$ we obtain

$$\mathbf{D}_{\dot\alpha\dot\beta} = \delta_{\dot\alpha\dot\beta}\int_{\gamma_m^+}\theta_{\dot\beta}\,\mathrm{d}S(\mathbf{x})\,,$$

which is a diagonal matrix.

Since γ_m^+ is the image of $(0,1)^2$ under a bilinear mapping (up to translation) and the surface mesh induced by $\mathscr{T}_{\ell_j}$ is a structured mesh, we can define the basis functions in terms of one-dimensional dual multiplier functions[172]. In the case that the parametrization

$$\varphi_m^+ : \mathbb{R}^2 \supset (0,1)^2 \longrightarrow \gamma_m^+ \subset \mathbb{R}^3$$

is an affine linear function we can define $\psi_{\dot\alpha}$ as $\psi^1_{\dot\alpha_1}\otimes\psi^1_{\dot\alpha_2}$, where $\dot\alpha = (\dot\alpha_1,\dot\alpha_2)$. On $(0,1)$, the one-dimensional Lagrange multiplier shape functions are given by

$$\hat\psi^1_0 = 2 - 3\cdot x\,, \quad \hat\psi^1_1 = 3\cdot x - 1\,.$$

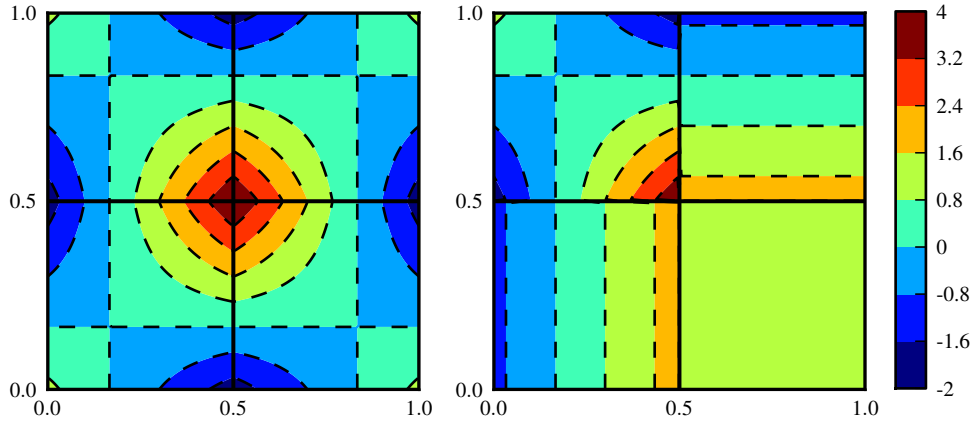


Figure 4.2. *Contour plot of the dual Lagrange multiplier function $\psi_{\hat{\alpha}}$. The left shows the basis function corresponding to an interior node. The right plot shows the basis function corresponding to the right lower corner of γ_m^+, i.e., the right and lower boundary of the shown rectangle are on $\partial\gamma_m^+$.*

On elements that touch the boundary of the non-mortar we set $\hat{\psi}_0^1 = 1$ or $\hat{\psi}_1^1 = 1$. In Figure 4.2 two dual Lagrange multiplier functions with support in the interior (left) and at the boundary of γ_m^+ (right) are plotted.

Since, in general, γ_m^+ is a curved surface with non-constant area element, we need to modify the basis functions to ensure that the biorthogonality condition (4.11) also holds for the transformed basis functions if the area element $\sqrt{\det\left((\nabla\varphi_m^+)^{\mathrm{T}}\nabla\varphi_m^+\right)}$ is not constant. We follow Flemisch and Wohlmuth [64] in rescaling the dual Lagrange multipliers by the inverse area element, i.e., we define

$$\psi_{\hat{\alpha}} = \frac{w_{\hat{\alpha}}}{\sqrt{\det\left((\nabla\varphi_m^+)^{\mathrm{T}}\nabla\varphi_m^+\right)}}\tilde{\psi}_{\hat{\alpha}} \circ \left(\varphi_m^+\right)^{-1}, \quad w_{\hat{\alpha}} = \frac{\int_{\gamma_m^+}\theta_{\hat{\alpha}}\,\mathrm{d}S(\mathbf{x})}{\int_{(0,1)^2}\hat{\theta}_{\hat{\alpha}}\,\mathrm{d}\mathbf{x}}. \tag{4.12}$$

Here, $\tilde{\psi}_{\hat{\alpha}}$ denotes the unscaled Lagrange multiplier defined on the structured mesh on $(0,1)^2$ not the multiplier functions on the reference element.

When assembling the matrix $\mathbf{R}$ on an interface Γ_{ij} we have to take into account the potentially different orientations of the structured meshes on the interface induced by $\mathscr{T}_{\ell_i}$ and $\mathscr{T}_{\ell_j}$. Because of this difference in the orientations, the parametrizations on the master and slave side can differ (up to an affine mapping) and thus it is not straightforwardly possible to transform the integral in equation (4.10b) to the reference element. In the Appendix A we discuss the assembly of $\mathbf{R}$ in detail. Note that the biorthogonality condition (4.11) is fulfilled independently of the relative orientation of slave and master faces.

4.3.4 Saddle-Point Formulation

One possibility to incorporate the mortar constraints (4.9) is to re-formulate the elliptic problem (4.1) as a saddle-point problem[16]. Hence, we search for a tuple $(V_\ell, \lambda_\ell) \in \mathbb{X}_\ell \times \left(\prod_{m=1}^{M} \mathbb{M}_{\gamma_m^+}\right)$ such that

$$\begin{aligned} a(V_\ell, U_\ell) + c(U_\ell, \lambda_\ell) &= b(U_\ell) \\ c(V_\ell, \mu_\ell) &= 0 \,, \end{aligned} \tag{4.13}$$

for all test tuples $(U_\ell, \mu_\ell) \in \mathbb{X}_\ell \times \left(\prod_{m=1}^{M} \mathbb{M}_{\gamma_m^+}\right)$ and

$$c(V_\ell, \mu_\ell) = \sum_{m=1}^{M} \int_{\gamma_m^+} [V_\ell]\, \mu_\ell \, \mathrm{d}S(\mathbf{x}) \,.$$

4.3.5 A Basis for the Subspace

An alternative approach to a saddle-point formulation is to construct a basis of the subspace $\mathbb{Y}_\ell^{\mathrm{m}}$ and assemble the stiffness matrix of a with respect to this basis. In this case, the elliptic nature of the problem is retained.

We obtain a basis of the constrained space $\mathbb{Y}_\ell^{\mathrm{m}}$ by eliminating the basis functions associated with nodes in the interior of the non-mortar side of each interface. More precisely, we categorize all mesh nodes as follows: The *inner nodes* are mesh nodes located in the interior of a patch Ω_i. *Master nodes* are mesh nodes located on a mortar or at the boundary of a non-mortar (i.e., on the wire basket). The remaining *slave nodes* are precisely mesh nodes in $\bigcup_{m=1}^{M} \mathcal{N}_{\gamma_m^+}^{\circ}$. Starting from the nodal basis $\boldsymbol{\theta}$ we construct a new basis $\boldsymbol{\pi}$ of $\mathbb{Y}_\ell^{\mathrm{m}} \subset \mathbb{X}_\ell$ as follows:

- Basis functions associated with inner nodes are not modified, i.e., $\pi_\alpha = \theta_\alpha$ for $\alpha \in \mathcal{N}_{\ell_i}^{\circ}$.
- Basis functions associated with slave nodes are dropped. The number of slave nodes is exactly the codimension of $\mathbb{Y}_\ell^{\mathrm{m}} \subset \mathbb{X}_\ell$.
- Basis functions associated with master nodes are modified in the following way: If $\alpha \in \mathcal{N}_{\gamma_m^-}$ and $\dot{\alpha} \in \mathcal{N}_{\gamma_m^+}^{\partial}$, we define

$$\begin{aligned} \pi_\alpha &= \theta_\alpha + \sum_{\dot{\beta} \in \mathcal{N}_{\gamma_m^+}} \mathbf{R}_{\dot{\beta}\alpha} \mathbf{D}_{\dot{\beta}\dot{\beta}}^{-1} \theta_{\dot{\beta}} \,, \\ \pi_{\dot{\alpha}} &= \theta_{\dot{\alpha}} + \sum_{\dot{\beta} \in \mathcal{N}_{\gamma_m^+}^{\circ}} \mathbf{C}_{\dot{\beta}\alpha} \mathbf{D}_{\dot{\beta}\dot{\beta}}^{-1} \theta_{\dot{\beta}} \,. \end{aligned}$$

  Since $\mathbf{D}$ is diagonal, π_α and $\pi_{\dot{\alpha}}$ have local support.

In shorthand notation we can write $\boldsymbol{\pi} = \mathbf{Q}^{\mathrm{T}}\boldsymbol{\theta}$ with

$$\mathbf{Q}^{\mathrm{T}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{P}^{\mathrm{T}} \end{bmatrix} .$$

Here, we ordered the degrees of freedom in $\mathbb{X}_{\boldsymbol{\ell}}$ and $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}$ so that inner nodes come first, then master nodes, and finally all slave nodes. For more details we refer to Maday et al. [110] and Bernardi et al. [26].

Because the basis functions in $\boldsymbol{\pi}$ are linear combinations of basis functions of $\mathbb{X}_{\boldsymbol{\ell}}$, we can easily express the stiffness matrix of a on $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}$ in terms of the block-diagonal stiffness matrix on $\mathbb{X}_{\boldsymbol{\ell}}$. A short calculation reveals that

$$\mathbf{A}^{\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}} = \mathbf{Q}^{\mathrm{T}}\mathbf{A}^{\mathbb{X}_{\boldsymbol{\ell}}}\mathbf{Q} . \tag{4.14}$$

We use equation (4.14) to implement the sparse matrix-vector multiplication by $\mathbf{A}^{\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}}$ without assembling the matrix. In fact, we only assemble $\mathbf{A}^{\mathbb{X}_{\boldsymbol{\ell}}}$ (which can be easily done in parallel on all patches) and implement multiplication by $\mathbf{A}^{\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}}$ in a matrix-free fashion. Due to the fixed bandwidth we can efficiently store and manipulate the product stiffness matrix. Since $\mathbf{A}^{\mathbb{X}_{\boldsymbol{\ell}}}$ can be decomposed as

$$\mathbf{A}^{\mathbb{X}_{\boldsymbol{\ell}}} = \bigoplus_{i=1}^{N} \mathbf{A}^{\mathbb{X}_{\ell_i}}$$

we moreover can locally reassemble $\mathbf{A}^{\mathbb{X}_{\boldsymbol{\ell}}}$ only on those patches where the level changes.

4.4 Linear Solver and Preconditioning

We use a preconditioned conjugate gradient solver in the constrained space $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}$ to solve the linear systems arising in the implicit-explicit Euler time discretization (Equation (4.1)) and the application of the L^2-transfer $\boldsymbol{\Pi}$ (Equation 4.16 in the next section).

For conforming discretizations, block Jacobi ILU preconditioners have proven to be efficient and exhibit good strong scalability in the number of subdomains, see Chapter 3. This motivates the use of the same preconditioning strategy for the adaptive method. However, a block decomposition of the stiffness matrix on the product space $\mathbb{X}_{\boldsymbol{\ell}}$ is insufficient as we have verified experimentally. Hence, we need to use a block decomposition of the basis $\boldsymbol{\pi}$ of $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}$. Here, again, explicit assembly of the local blocks of $\mathbf{A}^{\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}}$ is to be avoided as they feature a relatively high bandwidth in rows corresponding to master nodes and are cumbersome to handle efficiently with commonly used sparse matrix formats. Therefore, we prefer to refrain from constructing a local ILU decomposition.

Here, we propose to apply a fixed number of steps of a conjugate gradient solver to the local system $\mathbf{A}_i^{\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}}\mathbf{z} = \mathbf{r}$ starting from a zero initial guess. Our experiments have shown that a very small number of iterations (e.g., four) is optimal in terms of the time to solution for this preconditioner.

The sparse matrix-vector multiplication is implemented as follows. Considering the patch Ω_i, $1 \leq i \leq N$, let us write $\mathbf{V} = \begin{bmatrix} \mathbf{V}^{\mathrm{I}} & \mathbf{V}^{\mathrm{M}} \end{bmatrix}^{\mathrm{T}}$, i.e., we order the degrees of freedom such that interior

nodes come first. Then we can write the square sub-block of $\mathbf{A}^{\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}}$ corresponding to the degrees of freedom in Ω_i as

$$\mathbf{A}_i^{\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{C}^{\mathrm{T}}\mathbf{D}^{-1} & \mathbf{C}^{\mathrm{T}}\mathbf{R}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{A}_i^{\mathbb{X}_{\boldsymbol{\ell}}} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{\mathrm{op,SS}}^{\mathbb{X}_{\boldsymbol{\ell}}} \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{D}^{-1}\mathbf{C} \\ \mathbf{0} & \mathbf{D}^{-1}\mathbf{R} \end{bmatrix}, \tag{4.15}$$

where $\mathbf{A}_i^{\mathbb{X}_{\boldsymbol{\ell}}}$ itself is a 3×3 block matrix according to a decomposition of nodes into interior, master and slave nodes. By $\mathbf{A}_{\mathrm{op,SS}}^{\mathbb{X}_{\boldsymbol{\ell}}}$ we denote the slave-slave matrix entries on the non-mortar side opposite to the mortar face. Comparing equation (4.15) with equation (4.14) we find that applying $\mathbf{A}_i^{\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}}$ requires one additional sparse matrix-vector multiplication on the mortar side of patch faces.

4.5 Transfer Operators

The transfer of dynamic variables (membrane voltage and state variables) between two approximation spaces $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t)}$ and $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t')}$ at two different times t and t' is needed in Algorithm 4.1 to obtain a representation of the discrete solution in a new tailored approximation space.

The choice of mortar and non-mortar sides of an interface $\Gamma_{ij} = \Gamma_{ji}$ depends on the level and hence differs for $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t)}$ and $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t')}$. Consequently, even though the meshes $\mathscr{T}_{\ell_i(t)}$ and $\mathscr{T}_{\ell_i(t')}$ are nested for every patch Ω_i (with either one being the finer or the coarser) there is no simple relationship (in the sense of an interpolation operator or alike) between the basis functions $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$. In particular, patch-wise local transfer defines a mapping $\mathbb{X}_{\boldsymbol{\ell}(t)} \to \mathbb{X}_{\boldsymbol{\ell}(t')}$ that usually does not map $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t)}$ into $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t')}$.

In the context of a finite element discretization, the transfer via an L^2-projection is a natural choice. However, since the L^2-projection is a non-local operator, its evaluation requires the solution of a linear system for $(1+S)$ right-hand sides (one for each dynamic variable). For this reason we consider a second local transfer operator.

4.5.1 L^2-Transfer

We realize the transfer between the approximation spaces $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t)}$ and $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t')}$ by means of an L^2-orthogonal projection, ignoring the embedding into the product spaces for the time being. Hence, $\Pi\colon \mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t)} \to \mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t')}$ is characterized by

$$(\Pi V_{\boldsymbol{\ell}}, U_{\boldsymbol{\ell}})_{L^2(\Omega)} = (V_{\boldsymbol{\ell}}, U_{\boldsymbol{\ell}})_{L^2(\Omega)} \quad \text{for all } U_{\boldsymbol{\ell}} \in \mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t')} .$$

In contrast to a patch-wise local transfer operator, the L^2-projection Π is a global operator, the evaluation of which requires the solution of a linear system (unless $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t)} \subset \mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t')}$). Precisely, with respect to the bases $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$, Π is represented by

$$\boldsymbol{\Pi} = \mathbf{T}_1^{-1}\mathbf{T}_2 . \tag{4.16}$$

Here, $\mathbf{T}_1$ is the standard mass matrix on the space $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t')}$, i.e., the matrix representation of the L^2-scalar product with respect to basis $\boldsymbol{\pi}'$ and

$$(\mathbf{T}_2)_{\alpha\beta} = \int_\Omega \pi'_\alpha \pi_\beta \, \mathrm{d}\mathbf{x} \, .$$

Using the definition of $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$ in terms of the standard nodal basis functions, we can implement multiplication by $\mathbf{T}_1$ and $\mathbf{T}_2$ via multiplication of block matrices, see equation (4.14).

We use a (preconditioned) conjugate gradient method for solving $\mathbf{T}_1(\boldsymbol{\Pi}\mathbf{z}) = \mathbf{T}_2\mathbf{z}$. The size of this linear system depends on the number of dynamic variables (e.g., the number of gating variables and ionic concentrations in the chosen membrane model) and hence the solution can be expensive.

4.5.2 Local Transfer

To reduce the cost of the transfer operator, we consider the following alternative operator

$$\tilde{\boldsymbol{\Pi}} = \tilde{\mathbf{Q}}^{\mathrm{T}}\big(\textstyle\prod_{i=1}^{N} \mathbf{T}_3^i\big)\mathbf{Q} \, , \quad \tilde{\mathbf{Q}}^{\mathrm{T}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix} \, .$$

Here, $\mathbf{T}_3^i$ denotes a local interpolation or projection operator $\mathbb{X}_{\ell_i(t)} \to \mathbb{X}_{\ell_i(t')}$. The matrix $\tilde{\mathbf{Q}}^{\mathrm{T}}$ maps $\mathbb{X}_{\boldsymbol{\ell}(t')}$ to $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}(t')}$ by simply omitting slave values. We note that $\tilde{\boldsymbol{\Pi}}$ is a local operator similar to the interpolation and projection operators used in unstructured adaptive finite element methods.

4.6 Adaptivity Control

Our adaptive scheme is controlled by patch-wise accumulated error indicators and a maximum-based refinement strategy, as detailed in the following.

4.6.1 Error Estimation

We use the residual error indicator

$$\begin{aligned} \eta_E^2 = \mathcal{O}(\delta^2) &+ \sum_{\substack{F\subset\partial E \\ F\not\subset\partial\Omega_i}} \frac{\delta_F}{2g} \| \left[\mathbf{n}_E \cdot \mathbf{G}_{\mathrm{mono}} \boldsymbol{\nabla} V_{\boldsymbol{\ell}}^{i+1} \right] \|^2_{L^2(F)} \\ &+ \sum_{\substack{F\subset\partial E \\ F\subset\partial\Omega_i}} \frac{\delta_F}{g} \| \mathbf{n}_E \cdot \mathbf{G}_{\mathrm{mono}} \boldsymbol{\nabla} V_{\boldsymbol{\ell}}^{i+1} - \lambda_{\boldsymbol{\ell}} \|^2_{L^2(F)} \\ &+ \sum_{\substack{F\subset\partial E \\ F\subset\partial\Omega_i \text{ non-mortar side}}} \frac{g}{\delta_F} \| \left[V_{\boldsymbol{\ell}}^{i+1} \right] \|^2_{L^2(F)} \end{aligned} \tag{4.17}$$

where $E \in \mathscr{T}_{\boldsymbol{\ell}}$, $\delta_F = \mathrm{diam}(F)$ and g equals the largest eigenvalue of $\mathbf{G}_{\mathrm{mono}}$. $\lambda_{\boldsymbol{\ell}}$ denotes the Lagrange multiplier obtained as the residual on the slave side scaled by the matrix $\mathbf{D}^{-1}$, i.e.,

$$\lambda_{\boldsymbol{\ell}} = \sum_{\dot{\alpha}} \mathbf{D}^{-1}_{\dot{\alpha}\dot{\alpha}} \Big(b(\theta_{\dot{\alpha}}) - a(V_{\boldsymbol{\ell}}^{i+1}, \theta_{\dot{\alpha}}) \Big) \psi_{\dot{\alpha}} \, .$$

This estimator was introduced by Wohlmuth [170] and proven to be reliable and efficient on two-dimensional triangulations. Our implementation omits the higher-order volume term and uses midpoint quadrature (instead of high-order or exact quadrature) to reduce the cost of evaluating the error indicators.

Since we employ a time-window approach (see Section 4.1.1) we use accumulated error indicators for the refinement and coarsening. Moreover, since we mark patches (instead of individual elements) we are only interested in to the total estimated error per patch. Hence, refinement and coarsening is based on the accumulated errors $\left(\eta_i^\Sigma\right)_{i=1}^N$ where

$$\left(\eta_i^\Sigma\right)^2 = \sum_{E\in\mathscr{T}_{\ell_i}} \sum_{j=1}^{L_{\text{lap}}} \eta_E^2(t+j\cdot\tau) \,.$$

4.6.2 Marking Strategy

In this work we only consider marking strategies as used in multi-level finite element methods, in particular a maximum-based strategy which marks patches for refinement ($\ell_i \to \ell_i+1$) or coarsening ($\ell_i \to \ell_i-1$) based on the ratio $\eta_i^\Sigma / \max_j \eta_j^\Sigma$:

$$\ell_i \leftarrow \begin{cases} \ell_i+1 & \text{if } \eta_i^\Sigma \geq b\left(\max_j \eta_j^\Sigma\right) \\ \ell_i-1 & \text{if } \eta_i^\Sigma \leq a\left(\max_j \eta_j^\Sigma\right) \\ \ell_i & \text{else} \end{cases}$$

for some chosen values $0 \leq a < b \leq 1$. Hence, the patch level ℓ_i changes by at most one level in each mesh adaptation step. An average-based strategy is applicable as well but was found to be less efficient in early tests.

We use a weighted estimated error [101] for comparison against a given tolerance:

$$\frac{\left(\sum_i \left(\eta_i^\Sigma\right)^2\right)^{1/2}}{\text{atol} + \text{rtol}\cdot\|V_{\boldsymbol{\ell}}\|_{L^2(\Omega)}} \leq \text{tol} \,. \tag{4.18}$$

A mesh is accepted (i.e., no further passes are performed) if either (a) equation (4.18) is fulfilled, (b) no elements are marked for refinement or (c) the maximal number of repetitions is reached in Algorithm 4.1. Note that it might not be possible to satisfy equation (4.18), e.g., due to a bound on the level $\ell_i \leq \ell_{\max}$.

4.7 Implementation and Parallelization

One of the design goals of the presented adaptive scheme was the simplicity of the implementation and parallelization. In this section we comment on the structure of our reference implementation employed in Section 4.8.

4.7.1 Implementation Aspects

By design, the data structures required for the implementation of Algorithm 4.1 are straightforward extensions of data structures used in standard conforming finite element codes. In the implementation used in this work, we maintain an array of pointers (with fixed length N) pointing to instances of a structure storing

- the coefficients of the dynamic variables V, $\mathbf{s}$ with respect to the product space nodal basis;
- local product space mass and stiffness matrices;
- projection matrices $\mathbf{D}$, $\mathbf{C}$ and $\mathbf{R}$ for each of the six faces; as well as
- auxiliary (temporary) variables.

Note that since modifications to this data structure happen only during the regridding, we can avoid costly memory allocation and deallocation during the integration of a lap by keeping temporary arrays alive.

Only a minimal amount of metadata must be maintained. Each patch stores, for each of its neighbors, the index of the patch on the other side of the face, the level of this patch and the choice of the master/slave side.

The mass and stiffness matrix blocks on $\mathbb{X}_{\ell_i}$ have a fixed maximal bandwidth of 27 corresponding to the direct nearest neighbors. Hence, the multiplication can be efficiently implemented as a stencil operation with variable coefficients provided the input vectors are (at allocation time) padded with an additional halo layer. In comparison to other matrix storage schemes this approach is both SIMD (*single-instruction multiple-data*) friendly and memory efficient.

Similarly, we only store the coefficients of the projection matrices $\mathbf{C}$ and $\mathbf{R}$ on a face F and implement the matrix-vector product as a stencil operation. This is possible since the coarse-to-fine ratio on F is fixed (for at least one lap) and known prior to the assembly of the mortar projection. Since $\mathbf{D}$ is diagonal, it is stored as a vector on F.

In Algorithm 4.2, the implementation of the sparse-matrix vector product with a matrix $\mathbf{K}^{\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}}$ (which can be either the stiffness or the mass matrix) is described in detail. Note that we use the same storage location for the representation of a function $U_{\boldsymbol{\ell}} \in \mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}$ with respect to the basis of the subspace and with respect to the basis of the product space. We set the coefficients corresponding to eliminated basis functions θ_α to zero so that the application of $\mathbf{Q}$ and $\mathbf{Q}^{\mathrm{T}}$ are idempotent operations.

In general, the code needs to take into account the different relative orientations of neighboring elements that induce different orientations of the structured meshes on the interface. Our test implementation is build on the assumption of a structured coarse tessellation and therefore assumes matching orientations. We refer to Section 5.4 for a discussion of this problem in the context of a different implementation.

Note that our implementation does not use a numbering of the global degrees of freedom but only requires local patch-wise numbering of the mesh nodes. Due to the tensor structure of the mesh

1: **for** $i = 1, \dots, N$ **do** ▷ $U_\ell \leftarrow \mathbf{Q} U_\ell$
2: **for all** slave-side faces $F = \Gamma_{ji}$ of Ω_i **do**
3: Copy values of $U_\ell|_{\Omega_j}$ on F to a two-dimensional buffer B_1 ▷ Gather
4: Rotate B_1 entries according to the relative orientation of the patches ▷ not implemented (see text)
5: Copy values of $U_\ell|_{\Omega_i}$ on F to B_2 ▷ Gather
6: $B_3 \leftarrow \mathbf{D}^{-1}(\mathbf{R}B_1 + \mathbf{C}B_2)$
7: Replace $U_\ell|_F$ by B_3 ▷ Scatter
8: **end for**
9: **end for** ▷ U_ℓ is now represented with respect to the basis of $\mathbb{X}_\ell$

10: **for** $i = 1, \dots, N$ **do** ▷ $V_\ell \leftarrow \mathbf{K}^{\mathbb{X}_\ell} U_\ell$
11: $V_\ell|_{\Omega_i} \leftarrow \mathbf{K}^{\mathbb{X}_{\ell_i}}(U_\ell|_{\Omega_i})$
12: **end for** ▷ V_ℓ is now a function in $\mathbb{X}_\ell$

13: **for** $i = 1, \dots, N$ **do** ▷ $V_\ell \leftarrow \mathbf{Q}^{\mathrm{T}} V_\ell$
14: **for all** faces F of Ω_i **do**
15: **if** F is a master-side face **then**
16: Copy values of $V_\ell|_{\Omega_j}$ on F to B_1 ▷ Gather
17: Rotate B_1 entries according to the relative orientation of the patches ▷ not implemented (see text)
18: $B_2 \leftarrow \mathbf{R}^{\mathrm{T}}\mathbf{D}^{-1}B_1$
19: Add B_2 to $V_\ell|_F$ ▷ Scatter
20: **else** ▷ F is a slave-side face
21: Copy (interior) values of $V_\ell|_F$ to B_1 ▷ Gather
22: $B_2 \leftarrow \mathbf{C}^{\mathrm{T}}\mathbf{D}^{-1}B_1$
23: Add B_2 to $V_\ell|_F$ ▷ Scatter
24: Set interior value of $V_\ell|_F$ to zero
25: **end if**
26: **end for**
27: **end for** ▷ V_ℓ is now represented with respect to the basis of $\mathbb{Y}^{\mathrm{m}}_\ell$

Algorithm 4.2. *Implementation of the sparse matrix-vector multiplication $V_\ell = \mathbf{K}^{\mathbb{Y}^{\mathrm{m}}_\ell} U_\ell$ using the product space matrix $\mathbf{K}^{\mathbb{X}_\ell} = \bigoplus_{i=1}^{N} \mathbf{K}^{\mathbb{X}_{\ell_i}}$.*

$\mathscr{T}_{\ell_i}$ on Ω_i, a column-major or row-major ordering is a natural choice. In fact, in our implementation we make use of the support of multi-dimensional arrays in Fortran for storing and working with matrices and vectors.

The absence of a global numbering scheme allows us to perform local reassembly of the stiffness and mass matrices confined to patches where the level changes.

For fine level patches containing on the order of 16^3 or 32^3 elements, the proposed data structure naturally leads to "blocked" traversal of the degrees of freedom. In particular, the working set of the local block preconditioner discussed in Section 4.4 potentially fits into the level-three (or even level-two) cache of contemporary central processing units.

4.7.2 Parallelization

Our adaptive scheme permits parallelization using techniques well known in the finite element community. The presented parallelization scheme is optimized for moderately large systems (up to several hundreds of processing elements) but not for massively parallel processing, cf. Section 4.8.4.

For the parallelization of the method we use a non-overlapping decomposition of the coarse tessellation. Hence, each patch Ω_i is assigned to one processing element. The patch data structure discussed above can be reused without much modification. Only the metadata must be extended to store the owner processing element (identified by its rank) and the local index of neighboring patches.

An effective load balancing scheme must take into account weights $(w_i)_{i=1}^N \in \mathbb{R}^N$ that are assigned to each patch to account for the differences in computational intensity. A natural choice for these weights is the number of elements $\delta_{\ell_i}^{-3}$. In our implementation we augment this estimate for the load per patch by measured timings to improve the load estimation.

We use a Knapsack solver[131] to compute a well balanced partition of the coarse tessellation. This load balancing algorithm takes into account the weights but not the topology of the tessellation. Previous numerical experiments have shown that both, space-filling curve- and graph-based, partitioning algorithms perform significantly worse than Knapsack in terms of the achieved balance (and in consequence also lead to substantially worse scaling) due to the large coarse-to-fine ratios that lead to high variation in the weights.

Point-to-point communication is required for the repartitioning of the mesh, i.e., to exchange patch data when patches migrate, and in the implementation of the *mortar operations*, i.e., for the application of the operators $\mathbf{Q}$ and $\mathbf{Q}^{\mathrm{T}}$ used for mapping between constrained and product space.

In our implementation we exchange one message per patch per face. Messages can be distinguished by using the local patch index and face number as message tag. By storing the projection matrices $\mathbf{D}$, $\mathbf{C}$ and $\mathbf{R}$ for the same mortar on both processing elements we can tune the implementation to only communicate slave values (of which there are fewer on the interface) and hence reduce the communication volume. For example, in the computation of $\mathbf{Pz}$ we first apply $\mathbf{D}^{-1}\mathbf{R}$ locally on the master side and then send the result to the slave side. In the computation of $\mathbf{P}^{\mathrm{T}}\mathbf{z}$ on the other hand we first exchange the values of $\mathbf{z}$ and then apply $\mathbf{R}^{\mathrm{T}}\mathbf{D}^{-1}$ on the master side.

4.7.3 Measuring Depolarization Times

The depolarization time is an important observable in electrophysiological simulation. Usually it is defined as

$$t^{\text{depol}}(\mathbf{x}) = \min\{\,\text{time } t \mid \mathbf{s}_i(\mathbf{x},t) \geq q\,\}\ ,$$

for a given (membrane model-dependent) index $1 \leq i \leq S$ and threshold q. For the Bernus membrane model we use $i = 1$ (the m gate) and $q = 0.98$.

In practice, t^{depol} is measured at mesh nodes and interpolated between them. Since t^{depol} is not time-dependent, however, it is not possible to treat the depolarization time like other dynamic variables in a monodomain simulation. In particular, the approximation space $\mathbb{Y}^{\text{m}}_{\boldsymbol{\ell}(t)}$ is not well suited to approximate t^{depol} since $\mathbb{Y}^{\text{m}}_{\boldsymbol{\ell}(t)}$ will have by design low approximation quality far away from the depolarization front.

Here, we propose to use a second approximation space for the depolarization front. Since t^{depol} is an observable, we are flexible in the choice of the space. We use a product space

$$\mathbb{X}^{\text{depol}} = \mathbb{X}_{(\ell^{\text{depol}},\ell^{\text{depol}},\ldots,\ell^{\text{depol}})}$$

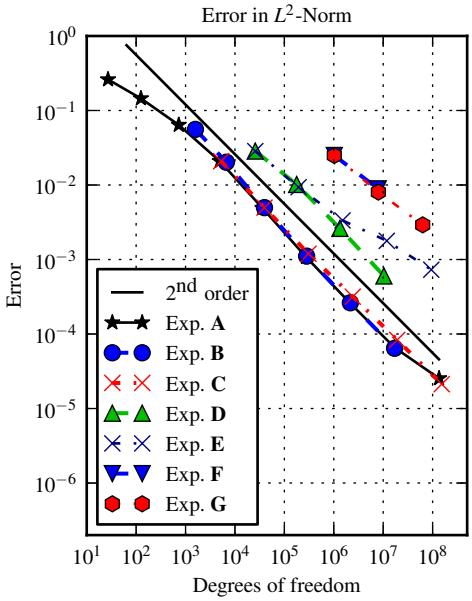
with a fixed level $1 \leq \ell^{\text{depol}}$ on all patches. The level ℓ^{depol} is chosen a priori.

In many cases, it is sufficient to capture t^{depol} on a coarser mesh than required for the computation of V and $\mathbf{s}$. However, even for moderate ℓ^{depol}, the memory required for storing t^{depol} can be a significant portion of the total memory usage, defying the purpose of an adaptive approach. Fortunately it is not necessary to hold all components of t^{depol} in memory at all times since cells depolarize only once during the depolarization phase. The array storing $t^{\text{depol}}\big|_{\Omega_i}$ is allocated on demand when the first mesh node in the patch Ω_i depolarizes and is removed from main memory (and written to disk) when all nodes in Ω_i are depolarized. In this way, memory has to be committed only for relatively few patches. The reduction of the memory usage achieved by this out-of-core implementation is directly proportional to the reduction in the number of degrees of freedom due to the adaptive strategy.

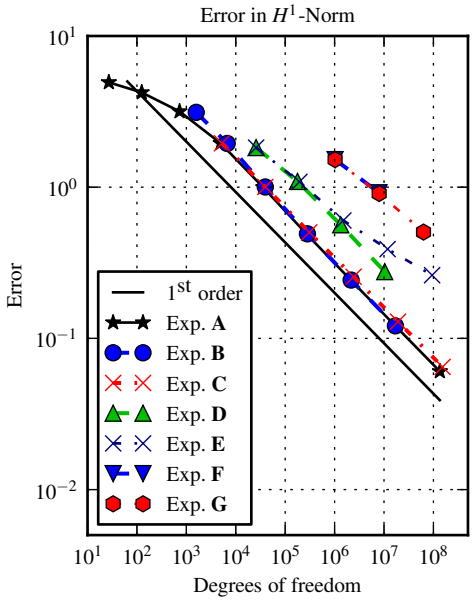
4.8 Results

The following tests have been performed on the Cray XE6 “Monte Rosa” at the Swiss National Supercomputing Centre, featuring dual-socket nodes with AMD Interlagos CPUs, 32 GiB main memory per node and a Gemini interconnect. To avoid a negative impact of the shared floating point units in the Bulldozer microarchitecture, in all experiments we placed only one process per Bulldozer module.

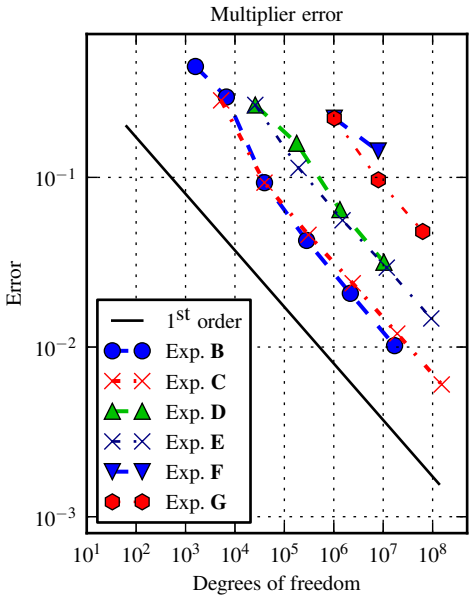
Our codes are written in Fortran 90 and compiled with the PGI 12.5-0 compiler.



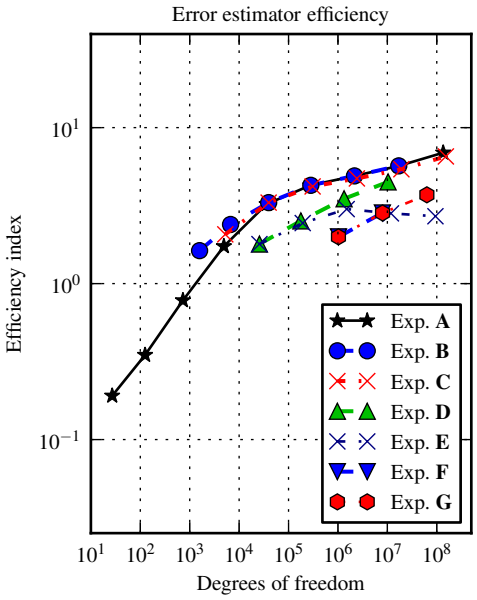
(a) *Convergence of the solution in the L^2-norm.*



(b) *Convergence of the solution in the H^1-norm.*



(c) *Convergence of the Lagrange multiplier in the mesh-dependent norm.*



(d) *Efficiency of the residual error estimator.*

Figure 4.3. *Error with respect to the exact solution and error indicator efficiency for Experiments **A** – **G**.*

4.8.1 Convergence Studies

In this section we analyze the convergence of the implemented mortar element method in the L^2- and H^1-norm as well as in the mesh-dependent multiplier norm

$$\|\lambda_\ell\|_\ell^2 = \sum_{\gamma_m^+} \sum_{\substack{F \text{ element} \\ \text{face} \subset \gamma_m^+}} \delta_F^{-1} \|\lambda_\ell\|_{L^2(F)}^2 .$$

For the numerical experiment, we considered the domain $\Omega = \left(\frac{1}{2}, \frac{3}{2}\right)^3$ and chose the right-hand side as well as Neumann values on $\partial\Omega$ such that the exact solution to the linear system in equation (4.1) is given by

$$V(\mathbf{x}) = \frac{2}{\pi} \operatorname{atan}\Big((|\mathbf{x}| - 1.732) \cdot 250 \Big) .$$

This solution features a steep wavefront similar to the solutions of the monodomain equation. For the following experiments we use $\mathbf{G}_{\text{mono}} = \mathbf{1}$ and $\tau = 0.05$.

We compare the results of following seven setups.

Experiment A. Conforming discretizations on a structured mesh of size 2^3 up to 512^3 elements.

Experiment B. Mortar discretization with 32 patches organized as a structured mesh of size 4^3 and 2^3 to 64^3 elements per patch.

Experiment C. Mortar discretization with 8^3 elements per patch and 2^3 up to 64^3 patches organized as a structured mesh.

Experiment D. Mortar discretization with 4^3 patches and a random assignment of levels $\ell \in \{1,2\}$. The number of elements per patch is increased from 2^3 on level one and 8^3 on level two up to 16^3 on level one and 64^3 on level two.

Experiment E. Mortar discretizations with 2^3 and 8^3 elements per patch on level one and two, respectively. The number of patches is increased from 4^3 up to 64^3. The assignment of levels $\ell \in \{1,2\}$ is chosen randomly for the 4^3-patches configuration. Patches in the finer tessellations inherit the level from the parent patch.

Experiment F. Mortar discretization with 4^3 patches and a random assignment of levels $\ell \in \{1,2,3\}$. The number of elements per patch is increased from 2^3 on level one, 8^3 on level two and 32^3 on level three up to 16^3 on level one and 64^3 on level two and 256^3 on level three.

Experiment G. Mortar discretizations with 2^3, 8^3 and 32^3 elements per patch on level one, two and three, respectively. The number of patches is increased from 4^3 up to 16^3. The assignment of levels $\ell \in \{1,2,3\}$ is chosen randomly for the 4^3-patches configuration. Patches in the finer tessellations inherit the level from the parent patch.

In Figures 4.3a–4.3c we show the norm of the error $V - V_\ell$ in the L^2- and H^1-norm as well as the error $\mathbf{n} \cdot \mathbf{G}_{\text{mono}} \nabla V - \lambda_\ell$ in the mesh-dependent norm $\|-\|_\ell$. When the number of patches is fixed,

we observe the expected second- and first-order convergence in the L^2 and H^1-norm, respectively, as well as first-order convergence of the Lagrange multiplier as predicted by the theory.

When increasing the number of patches with fixed number of elements per patch (Experiments **C**, **E** and **G**) we observe a reduced convergence order when the coarse-to-fine ratio is bigger than one. Note that in this case the number of degrees of freedom grows faster than the third power of the inverse mesh width and hence a sub-linear scaling can be expected.

Since the assignment of levels to patches is random in Experiments **D** – **G**, the distribution of patches with higher level (and, hence, finer local mesh) is not aligned with the position of the wave front. Therefore we observe a higher error for the same number of degrees of freedom in these experiments.

In Figure 4.3d we plot the ratio between estimated and actual error. For the conforming discretization we use a standard residual error estimator, omitting the first high-order term as we did in the definition of η_E in equation (4.17).

4.8.2 Small-Scale Problem

In this section we analyze the performance of the presented adaptive scheme for a model problem as used by Colli Franzone et al.[47]. We considered the domain $\Omega = (0,1)^2 \times (0, \frac{1}{16})$ (in units of centimeters) and fibers oriented in the xy plane with an angle of 45° with respect to the axes, i.e.,

$$\mathbf{a}_\mathrm{l} = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right]^\mathrm{T},\ \mathbf{a}_\mathrm{t} = [0,0,1]^\mathrm{T},\ \mathbf{a}_\mathrm{n} = \left[\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0\right]^\mathrm{T}$$

and

$$\mathbf{G}_\mathrm{mono} = 2 \cdot \mathbf{a}_\mathrm{l} \otimes \mathbf{a}_\mathrm{l} + 0.25562 \cdot (\mathbf{a}_\mathrm{t} \otimes \mathbf{a}_\mathrm{t} + \mathbf{a}_\mathrm{n} \otimes \mathbf{a}_\mathrm{n})\ \mathrm{mS/cm}\ .$$

We applied a stimulation current of $I_\mathrm{app} = 250\ \mu\mathrm{A/cm}^2$ for $\frac{1}{4}$ milliseconds in $(\frac{3}{8}, \frac{5}{8})^2 \times (0, \frac{1}{16}) \subset \Omega$. The coarse tessellation consisted of $16 \times 16 \times 1$ hexahedra ($N = 16^2$). We set $\ell_\mathrm{max} = 3$ and chose $(\delta_\ell)_{\ell=1}^3$ so that level one corresponded to 4^3 elements/patch, level two to 8^3 elements/patch and level three to 16^3 elements/patch. The mesh width on the finest level corresponded to a $256 \times 256 \times 16$ structured mesh on Ω. We used a time step size $\tau = 0.025$ ms, $L_\mathrm{lap} = 20$ and an absolute tolerance of 10^{-8} for the linear solver. The mesh adaptation was driven by the marking strategy described in Section 4.6.2 with parameters $a = 0.2$, $b = 0.5$, atol $= 10^{-3}$, rtol $= 1$ and tol $= 10^{-2}$. We compared our adaptive solution method to a structured grid solution method on a $256 \times 256 \times 16$ finite element mesh using a block Jacobi ILU method with 16 subdomains.

Figure 4.4a shows the depolarization time computed by the structured code. In Figure 4.4b we plot the difference of the depolarization time computed by the adaptive code relative to the result of the structured code. The adaptive scheme computes the depolarization time within a 4 % window. The peak of the relative difference is attained at the boundary of the activation site. In the remainder of Ω the difference is below $\sim 1.5\,\%$. In particular, the computation of the activation velocity is feasible with a small deviation. Note that, due to the differences in the discretization, a discrepancy of $\sim 1\,\%$ in the depolarization time is found between a $256 \times 256 \times 16$ conforming discretization and a mortar discretization using $16 \times 16 \times 1$ subdomains with structured 16^3 meshes per patch.

The relative spatial difference in the energy norm attains its highest value of $\sim 13.7\,\%$ during the depolarization phase (Figure 4.5).

In Figure 4.7 the execution time per lap is plotted for the adaptive and the structured code. The red curve shows the speedup (or slowdown) of the adaptive code relative to the structured code. While the adaptive code is $\sim 18.8\times$ faster during the repolarization phase, it does not achieve a speedup during the depolarization phase. In fact, during the first 20 ms of simulation time, the adaptive code is up to $\sim 6.3\times$ slower than the structured code. From the accumulated lap time shown in Figure 4.7 it is apparent that the overall execution time of the adaptive code during the depolarization phase from $t = 0$ to $t = 20$ ms is $2.3\times$ higher than the execution time of the structured code.

The number of iterations required per time step is $\sim 2\times$ higher for the adaptive code. Hence, since multiple repetitions of each lap are required, the total number of iterations per lap is up to $8\times$ higher. We note that restricting the number of passes would affect the accuracy of the method: In Figure 4.4c the relative error in t^{depol} for a simulation with a restriction of two on the number of passes is shown. In comparison with Figure 4.4b it is apparent that limiting the number of passes results in a relative error that is up to $3\times$ higher. Moreover, the numerical activation velocity is affected by this error.

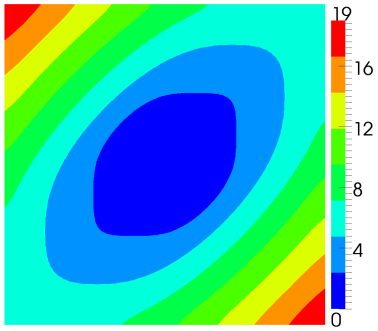
Comparison of L^2 and local transfer

The use of the local transfer operator $\tilde{\Pi}$ instead of an L^2-projection has only minor impact on the accuracy of the computed depolarization time, see Figure 4.4d. However, significant reduction in the execution time can be achieved. During the first 20 ms of simulation time, which are highly demanding for any adaptive approach, the reduction in execution time amounts to $\sim 28\,\%$. Compared to the structured code we therefore get a lower factor 1.8 (instead of 2.3).

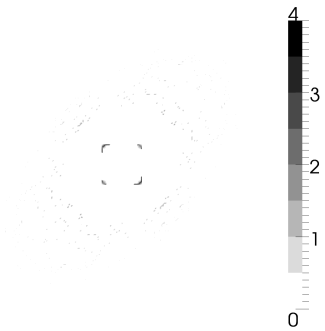
Comparison with Unstructured AMR

To better understand the performance of the presented method in comparison to other state-of-the-art approaches we implemented an unstructured adaptive monodomain solution method on conforming tetrahedral meshes. This code is based on the UG mesh manager[14] and PETSC[12]. For the simulation we used the same parameters as described above with the exception of $a = 2 \cdot 10^{-3}$ and $b = 5 \cdot 10^{-3}$ for the maximum-based marking strategy. A standard residual based error estimator for the Poisson equation was used. The initial mesh was obtained from a $16 \times 16 \times 1$ element structured mesh by subdividing each hexahedron into six tetraheda. The maximal number of refinements was set to four.

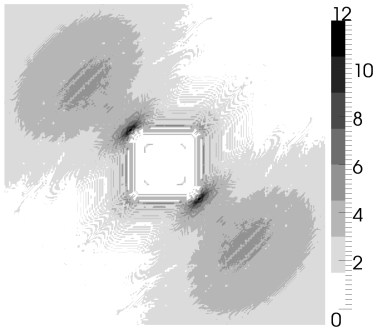
Figure 4.6 shows wireframe plots of the meshes constructed by the non-conforming method and by the unstructured AMR code at different stages of the simulation. The unstructured AMR algorithm captures the anisotropy of the solution better but it also requires refinement in a broader area around the depolarization front due to closures. During the simulation time $t = 5$ ms to $t = 15$ ms the non-conforming meshes consist of up to $3.38\times$ more mesh nodes. At the same time, the weighted estimated error measured by the non-conforming code is $2.4\times$ lower than the estimated error mea-



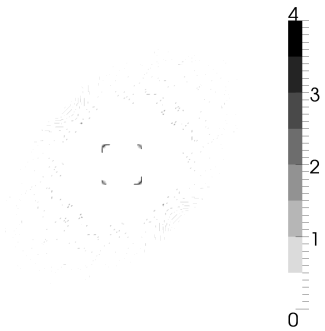
(a) *Depolarization time t^{depol} (in ms) computed by the structured code.*



(b) *Relative difference of adaptive code in percent.*



(c) *Relative difference of adaptive code with maximally two passes (in percent).*



(d) *Relative error of adaptive code with local transfer operators (in percent).*

Figure 4.4. *Depolarization times computed for the small-scale problem. A projected view of the domain $\Omega \subset \mathbb{R}^3$ is shown for clarity.*

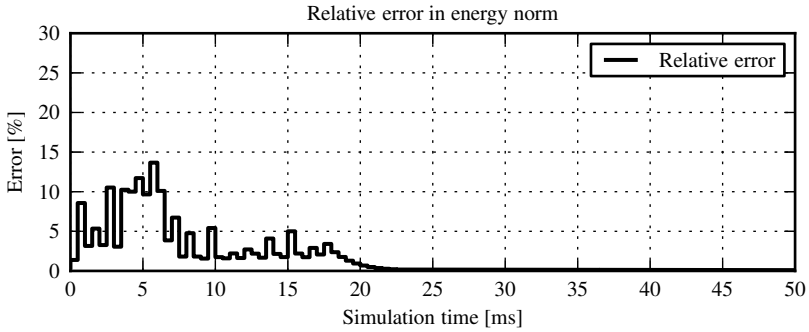


Figure 4.5. *Relative energy error of the adaptive method with respect to the result of the structured method. Shown is the spatial error at the end of each lap.*

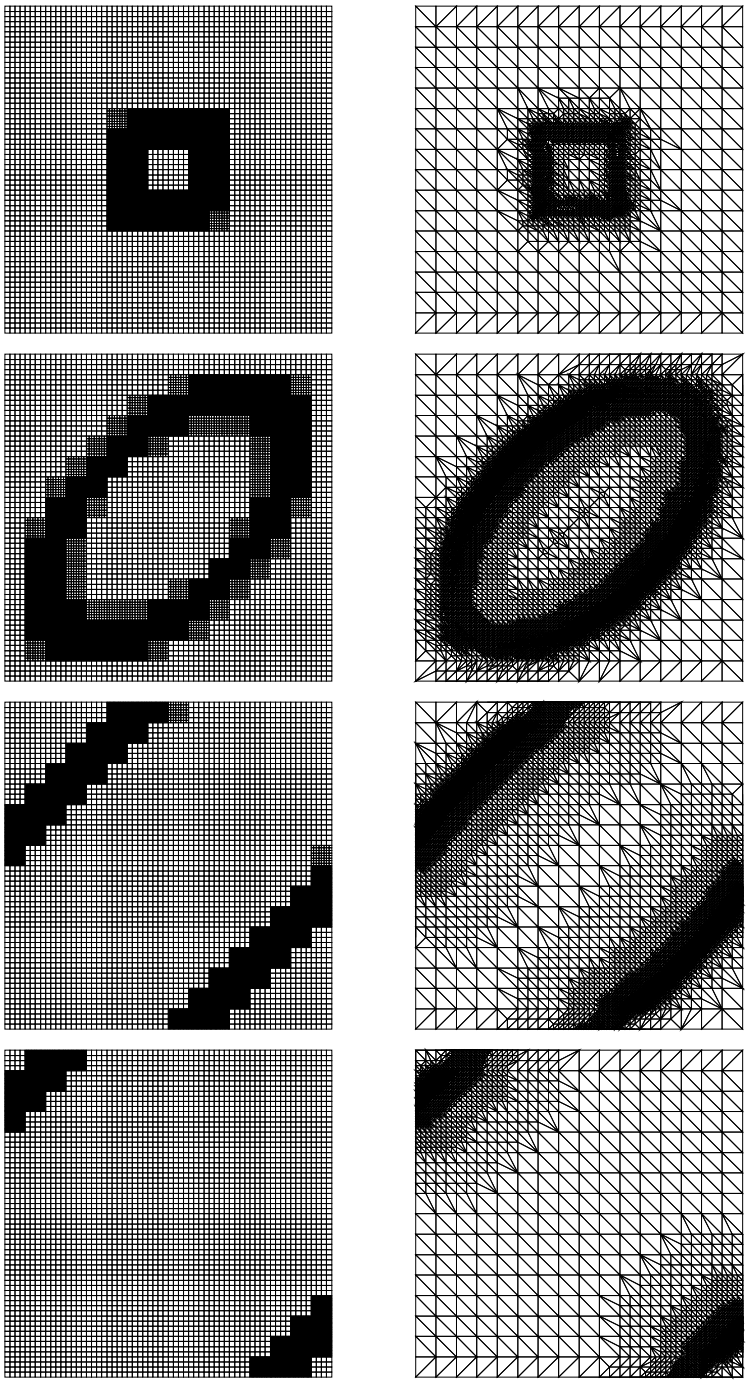


Figure 4.6. *Wireframe plot of the mesh $\mathscr{T}_{\boldsymbol{\ell}(t)}$ (left) and the unstructured adaptive mesh (right) at times $t = 0.5$, 5, 10, 15 ms for the small-scale problem.*

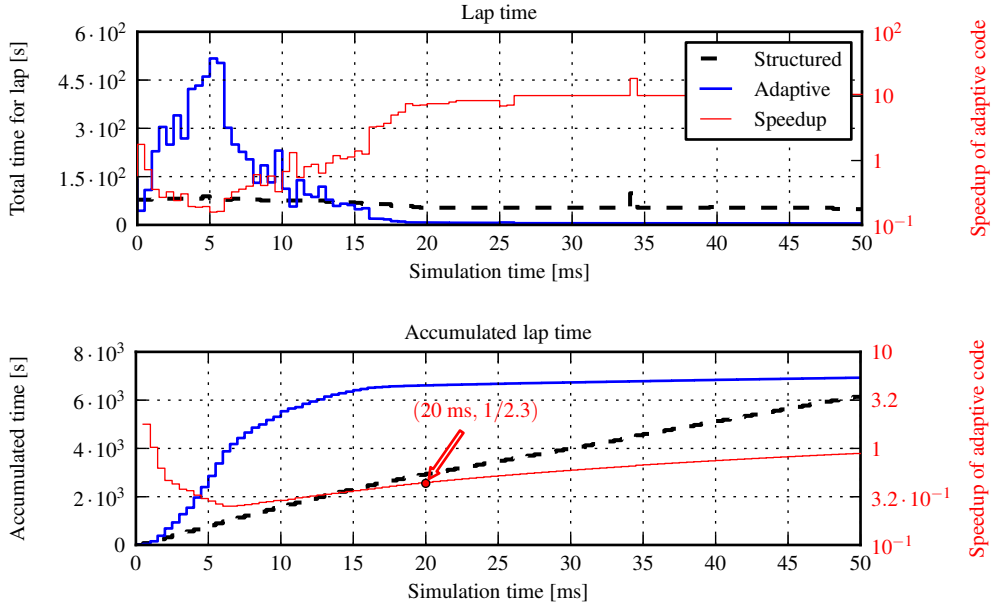


Figure 4.7. *Measured execution times for the small-scale problem. The upper graph shows the walltime for the execution of a lap of 20 time steps. Note that in the adaptive code each lap is repeated up to four times (cf. Figure 4.8). The lower plot shows the accumulated execution time.*

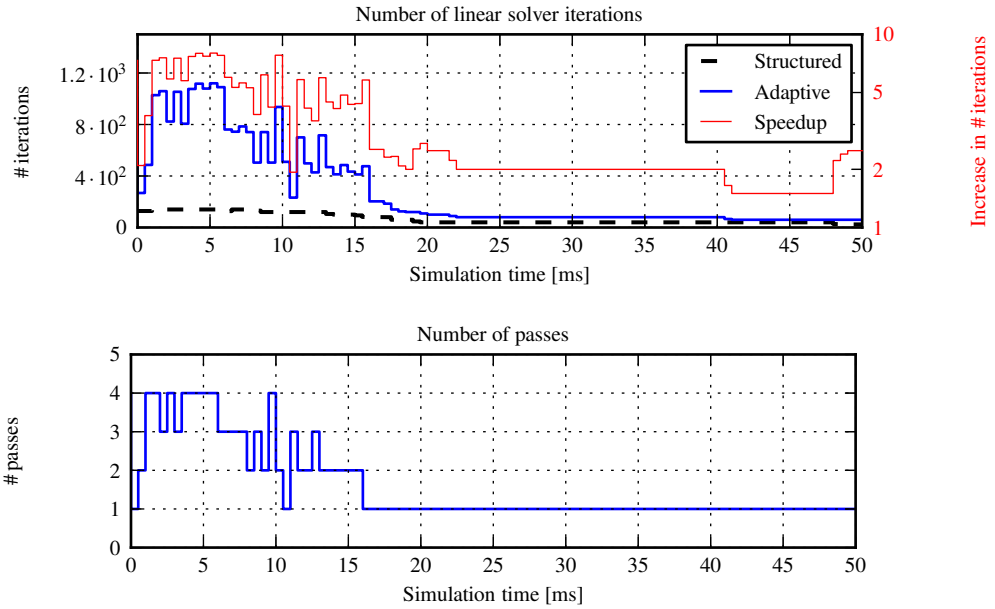


Figure 4.8. *Number of linear solver iterations per lap (upper plot) and number of passes for the integration of a lap (lower plot) for the small-scale problem.*

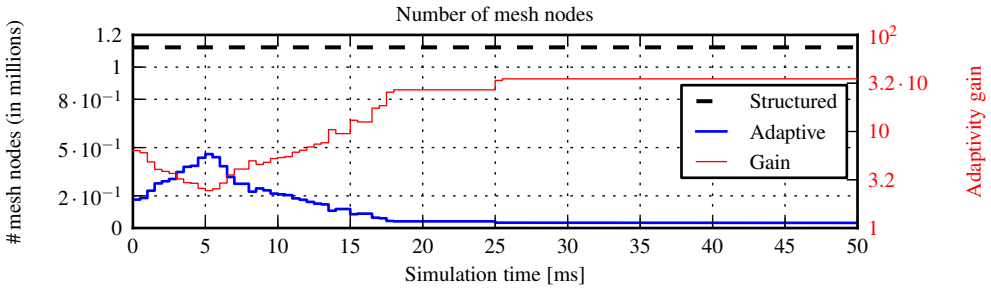


Figure 4.9. *Number of mesh nodes over time for the small-scale problem.*

sured by the unstructured AMR algorithm. Note, though, that the residual error estimators used in both algorithms feature different efficiency indices. The unstructured AMR algorithm requires on average 3.9× more passes over a lap. Even though we use an ILU preconditioner in our unstructured AMR method (instead of a block version as in the non-conforming adaptive method), we see an increase to up to 16 iterations per time step (compared to 15 for the non-conforming code), most likely due to the worsening quality of the finite element mesh.

Since our unstructured AMR implementation is not as well optimized as the non-conforming adaptive code we refrain from reporting execution times for this example.

4.8.3 Large-Scale Problem

In this section we analyze the performance of the proposed adaptive method on a demanding large-scale problem. We considered the model of a left ventricle, cropped at base and apex, as presented in Colli Franzone and Pavarino[46] with the same fiber orientations. The conductivity values were chosen as in Section 4.8.2; the geometry of Colli Franzone and Pavarino[46] was scaled by 1.4 to match it to previously applied models, cf. Potse et al.[127]. We applied a stimulation current of $I_{\text{app}} = 250\ \mu\text{A/cm}^2$ for 1 ms in the image of $(0.95,1) \times (0.125,0.175) \times (0.125,0.175)$ under the parametrization of Ω over $(0,1)^3$.

The coarse tessellation consisted of $4 \times 16 \times 32$ patches and we set $\ell_{\max} = 3$. We considered the choices $(\delta_\ell)_{\ell=1}^3 = (1/4, 1/8, 1/16)$ (setting **A**) and $(\delta_\ell)_{\ell=1}^3 = (1/2, 1/16, 1/32)$ (setting **B**). The mesh width on the finest level corresponded to a $64 \times 256 \times 512$ and $128 \times 512 \times 1024$ (67.1 million elements) structured mesh on Ω for **A** and **B**, respectively.

We used a time step size $\tau = 0.05$ ms, $L_{\text{lap}} = 20$ and an absolute tolerance of 10^{-8} for the residual norm in the linear solver. For the marking strategy we used the parameters $a = 0.01$, $b = 0.02$, atol $= 1$, rtol $= 0$ and tol $= 3$. The local transfer strategy from Section 4.5.2 was employed. We compare the adaptive method to a structured grid finite element method which used a block Jacobi ILU solver.

All simulations where run in parallel on 128 (adaptive) and 256 (structured) processing elements. We report timings as the sum of the time measured on each processing element. Under the assumption of ideal scalability, these timing results correspond to the serial execution time.

In Figure 4.14 the depolarization times computed by the structured code are shown. The computed membrane voltage V_ℓ and the adapted meshes at different steps are shown in Figure 4.15.

The measured execution times (Figures 4.10 and 4.11) show a similar picture as we obtained for the small-scale problem, i.e., while the adaptive procedure achieves significant speedup during the repolarization phase, this does not hold for the depolarization phase. During the first ~ 150 ms of simulation time, each lap is on average integrated three times, with some laps requiring four or five passes. At its minimum, the reduction in number of mesh nodes is found to be 4.4 and 5.2 for **A** and **B**, respectively.

As in Section 4.8.2, the number of linear solver iterations per time step is about two times higher for **A** than for the structured code. For **B**, however, the number of solver iterations is very high. In fact, for some time steps the linear solver did not reach the desired tolerance within 100 iterations (the maximal number specified).

Finally, for setting **A**, in Figure 4.13 we analyze the distribution of the execution time over the individual components of the algorithm. Up to $t = 150$ ms, the linear solver is a dominant component of the execution time. Evaluation of I_{ion}, integration of the state variables and the error estimation take only a small percentage of the computation time. During the depolarization phase, the majority of the compute time is spent for handling patches with $\ell_i = \ell_{\max} = 3$. Collective and point-to-point communication account for a large part of the execution time, in particular during the repolarization phase. The communication time itself is dominated by the time spent in collective communication (in particular, `MPI_Allreduce` calls in the linear solver).

4.8.4 Parallel Scalability

In this section we analyze the strong scalability of our implementation. The considered benchmark solves the monodomain equation on $\Omega = (0,1)^2 \times (0, \frac{1}{4})$ (in units of centimeters). We used the same fiber orientations as Pavarino and Scacchi [121] with the same conductivity tensor as in Section 4.8.2. A stimulation current of strength $I_{\text{app}} = 250\ \mu\text{A/cm}^2$ was applied for 1 ms in $(0, \frac{1}{8})^3 \subset \Omega$.

The coarse tessellation was a structured $16 \times 16 \times 4$ mesh on Ω. We set $\ell_{\max} = 3$ and chose $(\delta_\ell)_{\ell=1}^3 = (1/2, 1/8, 1/32)$, so that level one corresponds to 2^3 elements/patch, level two to 8^3 elements/patch and level three to 32^3 elements/patch. The mesh width on the finest level corresponded to a $512 \times 512 \times 128$ structured mesh on Ω. We used a time step size $\tau = 0.05$ ms, $L_{\text{lap}} = 10$ and repeated each lap up to nine times (nrep $= 9$). The mesh adaptation was driven by a maximum-based marking strategy (see Section 4.6.2) with $a = 0.01$ and $b = 0.1$.

In Figure 4.16a we plot the normalized execution times of the adaptive code for different laps. For comparison, the scaling of a monodomain solver on a structured $512^2 \times 128$ mesh is shown (this solution method used a block Jacobi ILU from the PETSC [12] package). The scaling of the adaptive code is good up to a certain number of processing elements (which depends on the lap) where execution time stagnates when adding additional processing elements. The fact that the execution time stays constant and does not increase thereafter indicates that the loss of scalability is associated with a poor load balance rather than, e.g., communication inefficiencies. This hypothesis

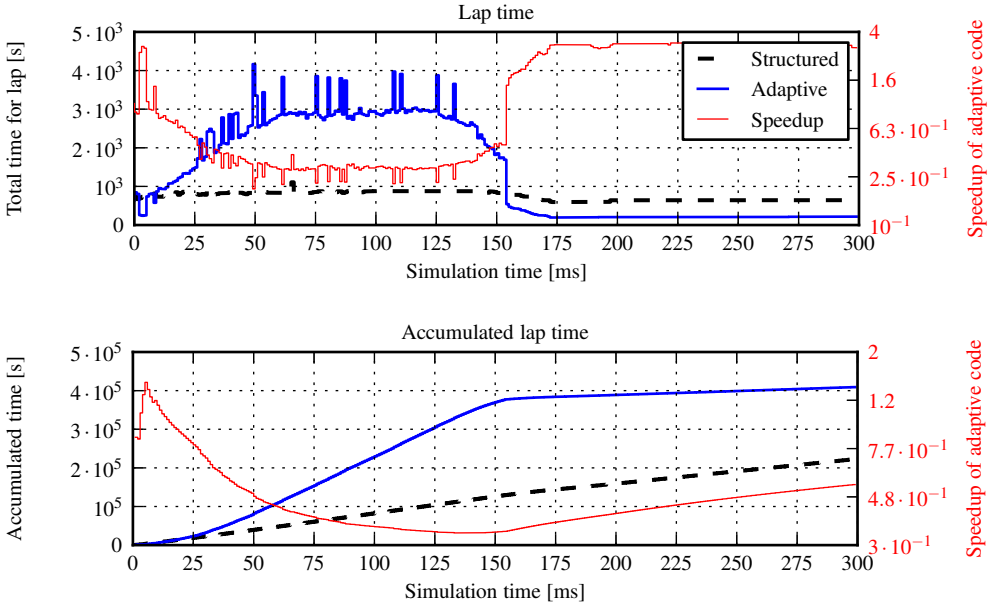


Figure 4.10. *Execution time of the adaptive code in comparison to a structured code for* ***A****. The upper graph shows the walltime for the execution of a lap of 20 time steps. The lower plot shows the accumulated execution time.*

is supported by Figure 4.16b which plots the imbalance measure

$$\frac{\max_{p=1,\dots,P} \#\text{elements assigned to } p}{\frac{1}{P}\sum_{p=1}^{P} \#\text{elements assigned to } p} = \frac{P \max_{p=1,\dots,P} \#\text{elements assigned to } p}{\#\text{elements}}$$

for different numbers of processing elements P. Comparing Figure 4.16a and Figure 4.16b it is obvious that the loss of scalability is directly related to the increase in load imbalance. By definition, a linear increase in the imbalance measure is equivalent to a constant maximal load. Since patches are assigned as a whole to processing elements and since the high coarse-to-fine ratios result in large differences in their costs (patches Ω_i with $\ell_i = 3$ are $\sim 4096\times$ more expensive than patches Ω_j with $\ell_j = 1$), scalability is lost as soon as the number of processing elements exceeds the number of patches on the finer levels.

Scalability is however not limited to 128 cores. For lap 27, the code scales up to 512 processing elements with an efficiency of 90.6% relative to 128 cores.

4.9 Related Work

Bernardi and Hecht [23], Bernardi and Maday [24] discuss the adaptive discretization of elliptic boundary value problems using a mortar element discretization in two spatial dimensions. In this work, mortar constraints are imposed on element edges where the triangular mesh is non-conforming. In

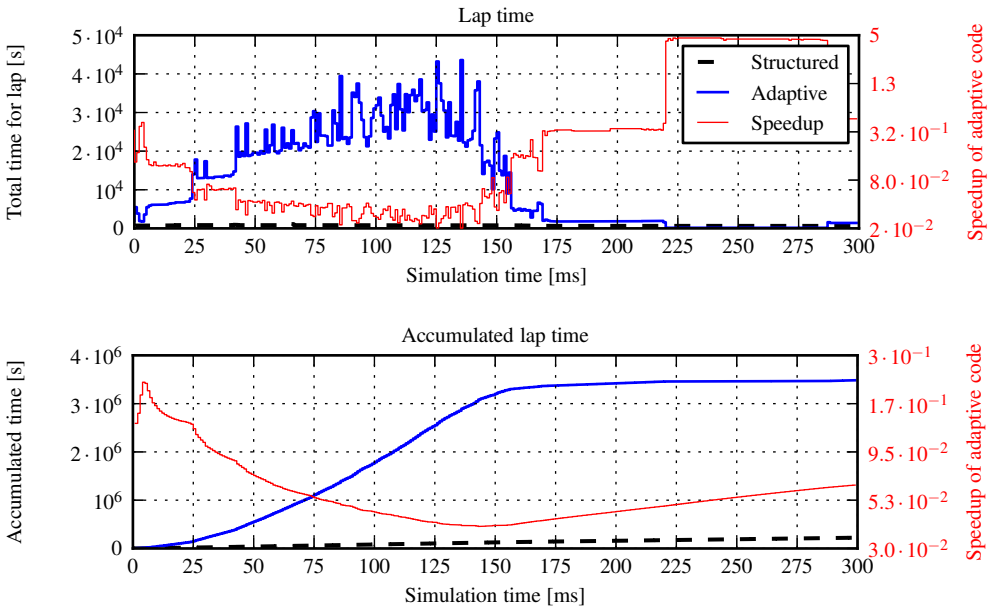


Figure 4.11. *Execution time of the adaptive code in comparison to a structured code for **B**. The upper graph shows the walltime for the execution of a lap of 20 time steps. The lower plot shows the accumulated execution time.*

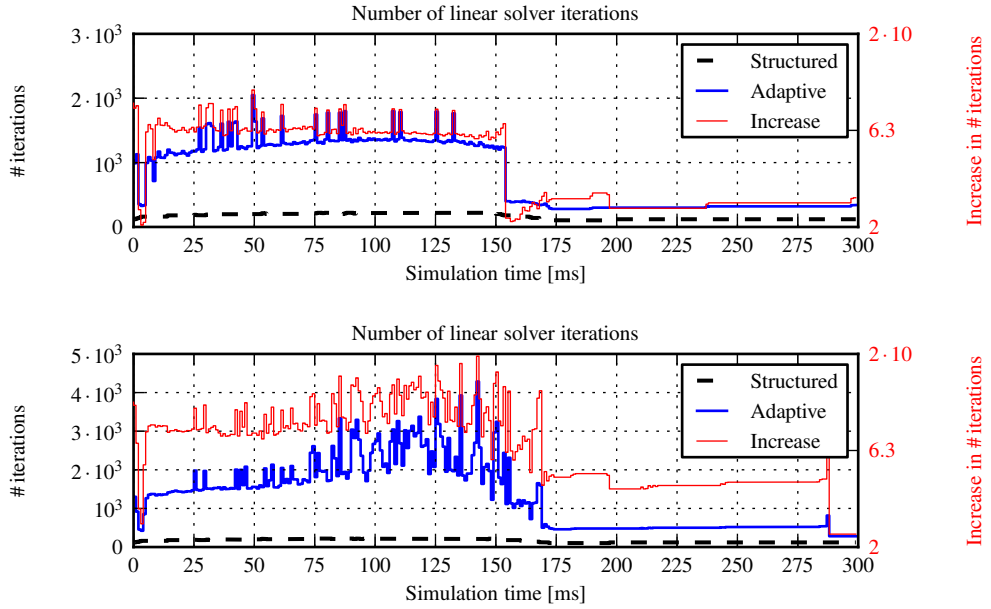


Figure 4.12. *Number of linear solver iterations for **A** (upper plot) and **B** (lower plot).*

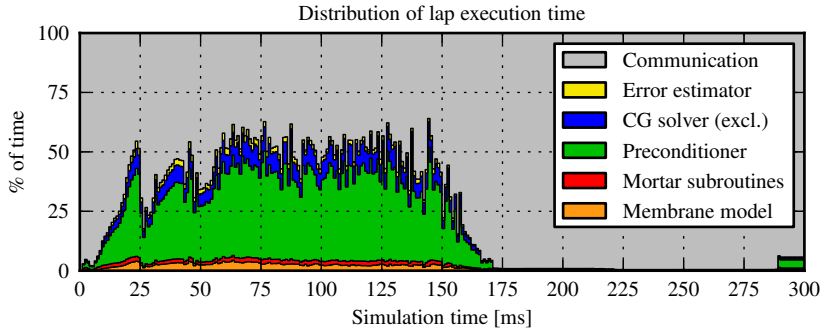


Figure 4.13. *Distribution of the execution time for problem **A**. The time measurements are summed over all passes over each lap.*

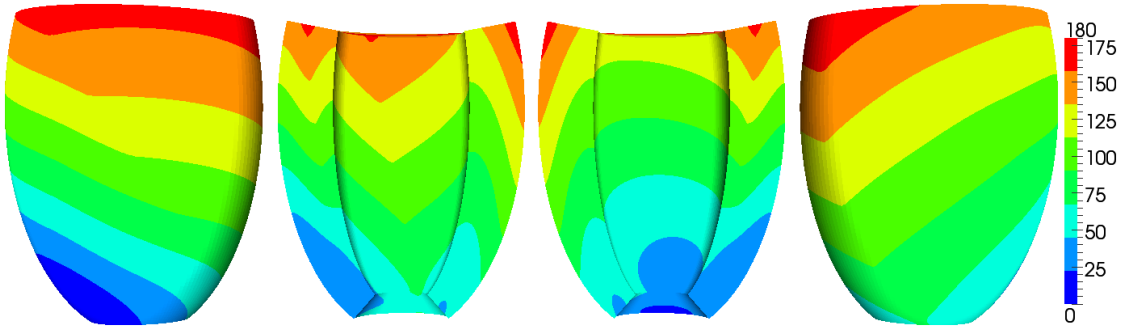


Figure 4.14. *Depolarization times t^{depol} (in ms) for the problem **A**. To simplify the visualization, the mesh has been downsampled by a factor four in each direction. The two plots on the right are rotated by 180° to visualize the back of the ventricle.*

contrast to this work, we use locally structured conforming meshes with weak constraints only on interfaces between elements of the coarse tessellation. Hoppe et al.[80] present an adaptive method that combines unstructured AMR with a mortar discretization. A hierarchical error estimator is used in this work. The authors present a domain decomposition solver for the saddle-point formulation of the discretized system and discuss the parallel implementation of the method. Feng et al.[63] discuss a three-dimensional mortar element method on geometrically non-conforming meshes. This study targets higher-order local approximation spaces and uses a matrix-free approach similar to ours. The parallelization of the method with OpenMP is discussed.

Linear solvers and preconditioners for mortar element discretizations have been studied by several authors. Abdoulaev et al.[1] discuss an iterative linear solver for the saddle point problem arising from a mortar element discretization. Bjørstad et al.[29] present a two-level additive Schwarz preconditioner for mortar elements. Braess et al.[32] and Wohlmuth and Krause[174] developed multigrid solvers for mortar element discretizations. Stefanica[147] studied the finite element tearing

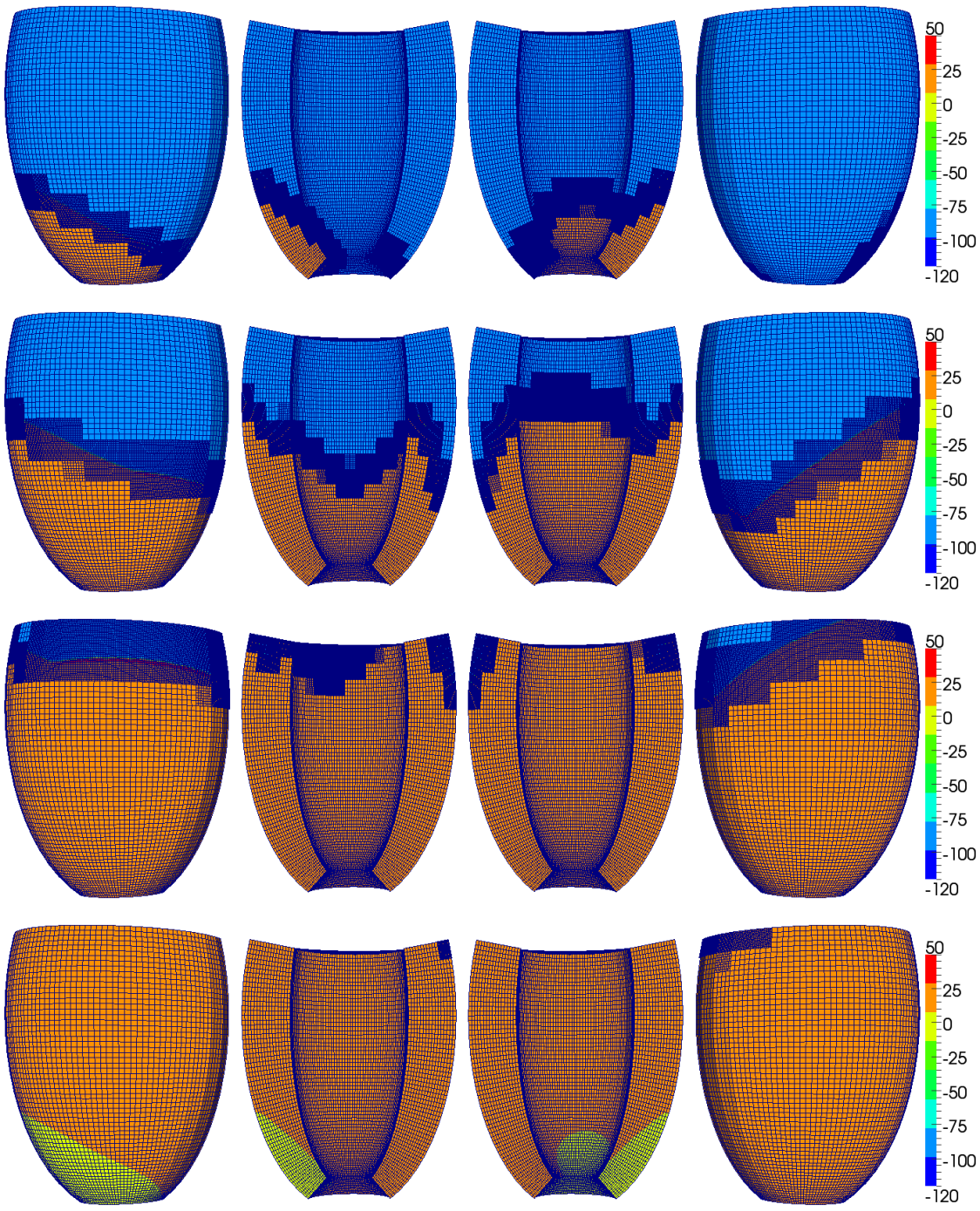
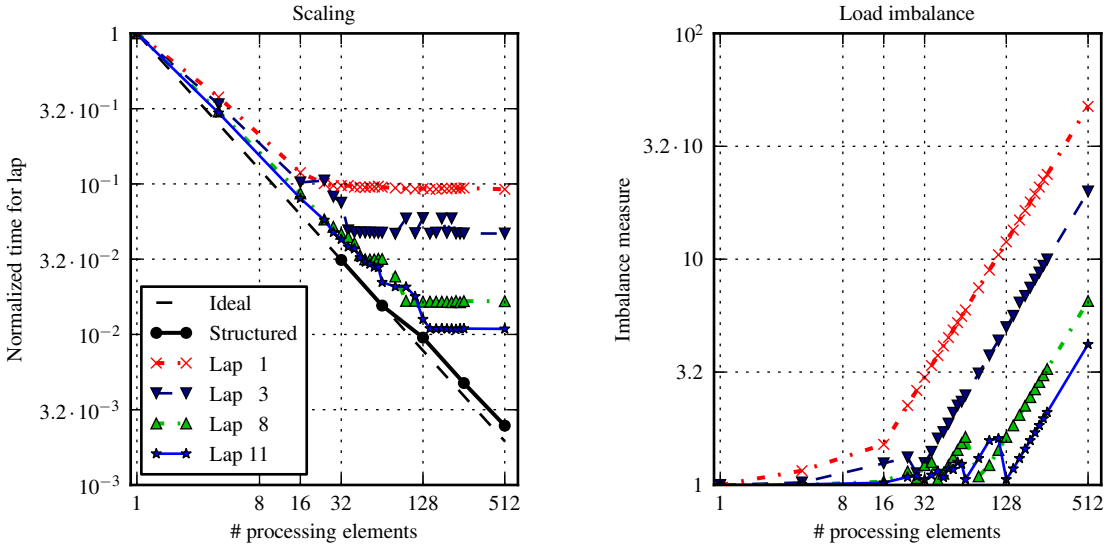


Figure 4.15. *Membrane voltage (in mV) and adaptive mesh at $t = 50, 100, 150, 200$ ms for **A**. The two plots on the right are rotated by 180° to visualize the back of the ventricle.*



(a) *Normalized execution times of the adaptive code (logarithmic scale) and, for comparison, of a structured monodomain solution method on a* $512^2 \times 128$ *mesh.*

(b) *Imbalance in the number of elements per processing element as a function of the total number of processing elements (logarithmic scale).*

Figure 4.16. *Strong scaling results.*

and interconnecting (FETI) method for the mortar element method. These works are usually concerned with ill-conditioned elliptic problems. For moderate coarse-to-fine ratios, we do not expect these methods to be competitive to our simple block preconditioner in terms of time-to-solution for the mass matrix dominated linear systems that we solve. However, our results show that block-preconditioning is not robust in the coarse-to-fine ratio and hence more complicated preconditioning techniques may be advantageous in this case.

The *asynchronous fast adaptive composite-grid* (FAC) method has been discussed by Hart and McCormick [74], Lee et al. [102,103], McCormick and Quinlan [112] for structured AMR with 2:1 coarse-to-fine constraints.

Memory-efficient data structures for adaptive mesh refinement have been studied, for example, by Bader et al. [9,10]. The authors consider the solution of the shallow water equation on adaptive triangular meshes using a discontinuous Galerkin discretization. The numerical solution is based on element-wise processing. Triangles are organized in a binary tree and ordered according to a Sierpinski space-filling curve. A stream- and stack-based system is used to organize the element-wise processing [8]. Weinzierl and Mehl [165] discuss a memory-efficient adaptive mesh solver using binary space partitioning trees.

A list of adaptive methods targeted at the monodomain or bidomain equation has been given in

Section 2.4.2. Here, we only compare our approach with those works that considered (semi-)implicit time discretization for three-dimensional problems.

Weiser et al. [166] reported that for a fibrillation study the employed adaptive scheme does not provide a reduction in the compute time despite a remarkable reduction of degrees of freedom by a factor of 150.

Belhamadia et al. [18] and Southern et al. [144,145] used anisotropic mesh adaptation to increase the accuracy of an unstructured finite element code. Speed improvements up to $13\times$ are reported for the solution of the bidomain equation in parabolic-elliptic form.

Ying and Henriquez [175] reported a $17\times$ speedup for a simulation of a dog heart ventricle. They used a second-order CBDF scheme for the phase-1 Luo-Rudy membrane model and a geometric multi-grid solver for the diffusion. Local time stepping was used for integrating the ordinary differential equations for the membrane model state variables. In our study, we have chosen a first-order time integration scheme that is currently most popular in computational electrocardiology [100,124]. For comparison, our structured code requires about $3.6\,\mu$s per time step per degree of freedom on a 2.1 GHz AMD Interlagos CPU. Based on the parameters given by Ying and Henriquez [175] their uniform grid solution method requires about $50\,\mu$s per time step per degree of freedom on a 3.6 GHz Intel Xeon. Moreover, we only consider spatial adaptivity to allow for an unbiased assessment of the efficiency of the non-conforming adaptive scheme.

In contrast to Ying and Henriquez [175] we used timings per time step/lap to assess the performance of the proposed method. In general, we do not consider end-to-end computing time a good measure for the efficiency of an adaptive scheme since a sufficiently long repolarization phase can mask potential inefficiencies of the adaptive scheme during the more interesting depolarization phase.

Only Southern et al. [145] have addressed the parallelization of their method and presented performance results up to 64 cores. Our implementation has been shown to scale up to 512 cores. None of the related studies have addressed the issue of computing depolarization times on adaptive meshes (see Section 4.7.3).

4.10 Discussion

We have proposed and investigated a novel adaptive scheme for reaction-diffusion equations based on a geometrically conforming mortar element method. The design goal was a method that is lightweight in the data structure, is relatively easy to implement and parallelize, and exhibits good performance on contemporary central processing units. In comparison to unstructured AMR and octree-based structured AMR, we choose a more complicated non-conforming discretization that allows us to simplify the mesh data structures. The method is based on patch-wise structured meshes encoded in a single vector $\boldsymbol{\ell} \in \mathbb{Z}^N_{\geq 1}$. Therefore, the memory required to store and modify meshes is minimal.

When assembled as a sparse matrix, the stiffness matrix on the mortar-constrained space has a high bandwidth at master nodes on interfaces where patches with fine and coarse local meshes intersect. Storing this matrix in standard formats (e.g., CRS or CCS), though possible, does not

allow for full exploitation of the structure of $\mathscr{T}_\ell$. For this reason we implemented matrix-vector multiplications in a matrix-free setup using stencil type operations. Motivated by the success of block Jacobi ILU preconditioning for conforming discretizations on uniform meshes (see Chapter 3) we have chosen a block Jacobi method for the adaptive scheme as well. By using a local CG solver for the preconditioner we remain in a matrix-free setup and obtain a very memory-efficient method.

The combination of a mortar element discretization with dual Lagrange multipliers and our matrix free setup appears to be well suited for the efficient implementation of a monodomain solution scheme on non-conforming meshes. In particular we would like to stress the advantages of the local nature of the mortar discretization, which imposes constraints only on degrees of freedom located in the interior of the interface between neighboring patches.

For the small-scale problem and problem **A** in Section 4.8.3, our method requires approximately twice as many iterations as an ILU or block Jacobi ILU preconditioner on a conforming structured mesh. In Figures 4.8 and 4.12 the total number of iterations (taking into account multiple repetitions) are shown. The results for setting **B** in Section 4.8.3 indicate that the iteration numbers are influenced by the coarse-to-fine ratio. However, it is well known that also for conforming discretizations and, e.g., multi-grid solvers, spatial adaptivity can have a negative impact on the solver efficiency. In fact, the ILU preconditioned CG used in the unstructured AMR algorithm required more iterations than the linear solver in the non-conforming algorithm, as reported in Section 4.8.2.

To reduce the overhead due to adaptivity, we have applied two optimization techniques in this study. First, we used a low-order quadrature to approximate the residual error estimator (Section 4.6.1). The error in the estimated error due to this modification is within a few percent. Moreover, we have introduced a local transfer operator (Section 4.8.2) which can be used as a drop-in replacement for the L^2-transfer.

The proposed parallelization scheme has been shown to be effective up to several hundreds of cores (Section 4.8.4). When the number of patches on the finer mesh levels is too low compared to the number of processing elements, maintaining the load balance may become difficult. As one expects in an adaptive scheme with varying number of degrees of freedom, parallel efficiency varies over the course of a simulation. A hybrid MPI+threads implementation (e.g., using OpenMP for loop-level parallelism) might be used to improve the scaling at larger core counts.

The design of adaptive numerical algorithms necessitates a trade-off between computational efficiency and the "optimality" of the constructed approximation spaces. The presented method represents an edge case in this spectrum of adaptive methods, as it vigorously favors efficiency of the data structures over a reduction in the degrees of freedom. This choice has two important consequences that can be observed in our experiments. On the one hand, we measure a relatively low reduction in the degrees of freedom compared to other unstructured and structured AMR methods. Considering a two-level method with coarse-to-fine ratio r, a back-of-the-envelope calculation shows that r^3 is an upper bound for the reduction in the degrees of freedom achieved if the depolarization front is an axis-aligned hyperplane and the coarse tessellation is a sufficiently fine structured mesh. For setting **A** in Section 4.8.3 this means that the adaptivity gain is bounded by $4^3 = 64$. For

a more complicated shape of the depolarization front, the gain by adaptivity will be smaller. This is in fact what we observe. For **A**, the reduction in the number of mesh nodes is only about $4.4\times$ at its minimum. On the other hand, the number of repetitions required to find a tailored mesh (given a desired error tolerance and a bound on the maximal level) is low compared to, e.g., an unstructured AMR algorithm (cf. Section 4.8.2). In all our experiments, our marking strategy terminated within 3–5 passes over a time window. Since the mesh $\mathscr{T}_{\boldsymbol{\ell}}$ is encoded by the vector $\boldsymbol{\ell}$ it is possible to adapt the mesh to the solution disregarding the refinement history. This could be used to develop more effective problem-tailored marking strategies that would further speed up the adaptive method.

In the presented numerical experiments we have compared our adaptive method to a state-of-the-art structured solver for the monodomain equation. The observed net slowdown of the adaptive code relative to a structured code that we measure in Section 4.8.3, is explained by the combination of a comparably low reduction in the degrees of freedom, the higher solver cost ($2\times$), the need to repeat laps multiple times, additional overhead (data transfer, matrix reassembly, error estimation) and differences in the parallel scalability. An improved preconditioner and marking strategy might help to narrow or close this gap. Let us point out that all comparisons have been made between the adaptive strategy and a *structured* uniform monodomain code, which typically outperform *unstructured* uniform monodomain codes that are most relevant for practical applications.

As is the case for any adaptive method, in order to deal with complex geometries, a suitable coarse tessellation has to be constructed. For complicated geometries as obtained from medical imaging, the construction of a suitable coarse tessellation as used in Section 4.2 might be difficult. Another aspect is that many models for heart tissue feature jumps in the coefficients and use different membrane models in different regions. Also these heterogeneities have to be to be taken into account when constructing the coarse tessellation.

In the next chapter we will consider a variant of this lightweight adaptive scheme which uses a forest of shallow trees to manage the local tensor meshes instead of the coarse tessellation employed in this chapter. The goal of this approach is to address the challenges discussed above by increasing the flexibility of both the adaptive mesh data structure and the solver technique.

5 Spatial Adaptivity Using Forests of Shallow Trees

In the previous chapter we have proposed a lightweight scheme for solving the monodomain equation on adaptive meshes. The goal of this chapter is to extend this method in two important ways. First, we want to address the challenges faced in the practical application of the lightweight adaptive scheme, namely the relatively low reduction in the number of degrees of freedom and the scalability limits. Moreover, we want to extend the approach to a broader class of partial differential equations and to different time discretization schemes.

In this chapter we present the idea of using forests of shallow trees for spatial adaptivity. We discuss the construction of ansatz spaces on the resulting mesh data structures and our strategy for the assembly of mass and stiffness matrices. We present the results of numerical experiments and demonstrate the potential of the method for large-scale heart models.

5.1 Introduction

For the design of adaptive methods several factors must be taken into account and weighted according to importance. For example, one wants to maximize the reduction in the degrees of freedom while minimizing the overhead (due to data structures or required repetitions of the adaptation process) at the same time. In this chapter we propose an extension of the lightweight adaptive scheme from Chapter 4 which is based on a slightly more expensive mesh data structure (based on shallow trees) and a more flexible implementation of the linear algebra procedures.

The design goals of the presented method were strongly influenced by the results from Section 4.8. First, we aimed at a method that could achieve a larger reduction in the degrees of freedom than what can be achieved in practice with the mesh data structure from Chapter 4. Second, the method should address the scalability limits observed in Section 4.8.4. Third, in order to be able to use different time integration schemes and solve a larger variety of equations, the method should be based on a more flexible set of linear algebra routines.

We address the first two design goals by reducing the size of the basic building blocks of our adaptive schemes. In Chapter 4 a patch Ω_i was the basic building block of the discretization as

well as the load balancing. Here, we use the leaves of a 2^d-tree (see Section 5.2) as the basic building block. This allows for a more precise placement of the refined regions and thus increases the adaptivity gain. Moreover, it increases the number of entities that are available to the load balancing scheme.

To increase the flexibility of the method we assemble stiffness matrices directly on the subspaces, rather than using a matrix-free approach as in Chapter 4. On the one hand, using this approach we cannot take advantage of the structure of the product space stiffness matrices for efficient storage and local reassembly. On the other hand, however, it allows for a flexible choice of preconditioners, including black box solvers such as an algebraic multi-grid.

5.2 Adaptive Meshes on Forests of Shallow Trees

In Section 4.2 we proposed a lightweight data structure for adaptive meshes which was based on a fixed tessellation $\Omega = \bigcup_{i=1}^{N} \Omega_i$ and a vector $\boldsymbol{\ell} \in \mathbb{Z}_{\geq 1}^N$ that assigned a level to each patch Ω_i. Advantages of such a simple data structure are the minimal storage requirements and the great flexibility (due to a complete lack of "history") that allows to resolve strongly varying behavior in time. However, as discussed in Section 4.10, the adaptivity gain that can be achieved using these meshes is limited by the width of the elements in the coarse tessellation. When dealing with "sharp" localized features (e.g., wave fronts) it is desirable to have more control over the shape and position of the refined regions.

One possible path to facilitate this fine-grained control is to adopt the approach of block-structured adaptive methods by replacing the single structured mesh $\mathscr{T}_{\ell_i}$ on the patch Ω_i by a set of floating rectangular blocks, the positions of which are prescribed as integer coordinates with respect to nested coordinate systems on Ω_i. Here, we follow a different approach using a forest of trees approach as described by Burstedde et al. [35,36,37,38]. Hence, we assign a 2^d-tree (binary tree, quadtree, octree, or 16−tree) to each patch Ω_i. The leaves of these trees define the blocks to which we assign a structured mesh based on the level. Our approach differs from the one by Burstedde et al. [38] in two important aspects. First, we consider tree leaves as blocks (to which a structured mesh is assigned) instead of elements. Due to the implicit structure of block meshes, this results in a compactification of the data structure and should benefit a more flexible handling of the mesh. Second, since leaves correspond to batches of, e.g., 4^d or 8^d elements, our focus is on *shallow* trees with only few levels.

As in Section 4.2, we assume that Ω_i is equivalent to $(0,1)^d$ up to translation and a linear, bilinear, trilinear or quadlinear mapping for $d = 1,\dots,4$, respectively. We provide a purely integer-based representation of the tree. Geometrically, the corner coordinates of tree leaves can be transformed from $[0,1]^d$ to $\overline{\Omega_i}$ using the transformation.

A 2^d*-tree* is obtained by recursively splitting axis-parallel boxes into 2^d sub-boxes starting with $[0,1)^d$ as the root. Here, we only consider *complete* 2^d-trees which have the property that each node in the tree either has 2^d children or none (i.e., is a leaf). In such a tree, the coordinates of a child box with respect to the parent node can be described by a vector $\mathbf{i} \in \{0,1\}^d$ or equivalently (using a lexicographical ordering of the set $\{0,1\}^d$) by a number $0 \leq i \leq 2^d - 1$. Note that the entries of $\mathbf{i}$

equal the digits in the radix-2 representation of the number i. A leaf in the tree is uniquely identified by its level ℓ and the sequence $i_1, i_2, \ldots, i_\ell$ of relative coordinates. Assuming a maximum level of $15 = 2^4 - 1$ we can combine these natural numbers into a single number

$$o = \sum_{j=1}^{\ell} i_j \cdot 2^{4+d(\ell-j)} + (\ell - 1) \in \mathbb{Z}_{\geq 0}$$

that uniquely defines the tree leaf. o is called the *Morton index* of the leaf and the induced ordering of tree leaves is called *Z-ordering* (see, for example, Gaede and Günther[66]). Note that o can be stored as a 32-bit integer if $\ell \leq 15, 14, 9, 7$ for $d = 1, 2, 3, 4$, respectively. Since a 2^d-tree is described completely by its leaves we can use a *linear* storage and only store the Morton indices of the leaves. Thus we can identify a tree with an element of $(\mathbb{Z}_{\geq 0})^*$. Note though that not every element of $(\mathbb{Z}_{\geq 0})^*$ defines a valid, complete 2^d-tree. In Figure 5.1 an example for a 2^2-tree is shown.

We assign a tree $\tau_i \in (\mathbb{Z}_{\geq 0})^*$ to each patch Ω_i. On each leaf o of level ℓ in τ_i, a structured mesh $\mathscr{T}_o$ is induced by transforming a structured mesh of width δ_ℓ from $[0,1)^d$ to the hexahedron in Ω_i that is represented by o. Combining these local structured meshes we obtain the *mesh* $\mathscr{T}_{\boldsymbol{\tau}}$ which is uniquely defined by the choice of $\boldsymbol{\delta}$ and the vector $\boldsymbol{\tau} = (\tau_i)_{i=1}^N$. Note that the construction of $\mathscr{T}_{\boldsymbol{\tau}}$ is completely analogous to that of $\mathscr{T}_{\boldsymbol{\ell}}$ in Section 4.2. In Figure 5.2 the construction of $\mathscr{T}_{\boldsymbol{\tau}}$ is illustrated (compare to Figure 4.1).

5.3 Discretization

Similar to the definition of the product space $\mathbb{X}_{\boldsymbol{\ell}}$ in Section 4.3 we can define a product space

$$\mathbb{X}_{\boldsymbol{\tau}} = \prod_{i=1}^{N} \prod_{o \in \tau_i} \mathbb{X}_{\tau_i, o}$$

where $\mathbb{X}_{\tau_i,o}$ denotes the local approximation space on the leaf $o \in \tau_i$ built using first-order finite elements. As proven in Section 4.3 the linear space $\mathbb{X}_{\boldsymbol{\tau}} \not\subset H^1(\Omega)$ is not a suitable approximation space for use in a Galerkin method. Instead, we construct suitable subspaces that serve as ansatz and test spaces in a Galerkin method.

5.3.1 Geometrically Non-Conforming Mortar Discretization

In Section 4.3 we discussed the definition of the mortar approximation space $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\ell}}$ build on top of the mesh $\mathscr{T}_{\boldsymbol{\ell}}$. A similar approach allows for defining a non-conforming approximation space $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\tau}}$ on top of $\mathscr{T}_{\boldsymbol{\tau}}$. In comparison with the method detailed in Chapter 4 the construction of $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\tau}}$ is complicated by the fact that the decomposition of Ω into tree leaves is generally not conforming, see Figure 5.3a. Thus, a non-mortar γ_m^+ can correspond to multiple mortars and in particular the number of mortars and non-mortars differs. We associate the mortar side of a face to the leaves with the lowest level or, in case the level is equal, to the leaf with the lower key. Due to the hierarchical tree structure it is guaranteed that the choice of mortar and non-mortar sides is made consistently. In the following

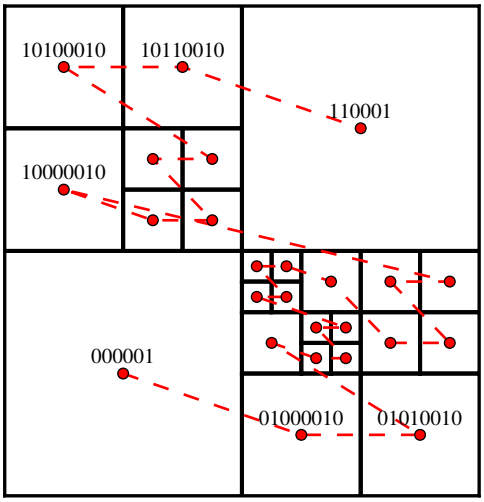
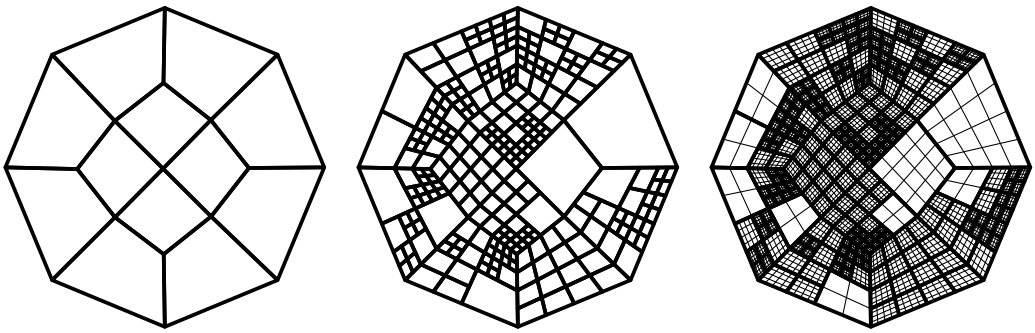


Figure 5.1. *Sketch of a quadtree. The leaves are ordered by their Morton index starting at the left lower leaf with key 000001. For leaves with level ≤ 2 the binary representation of the Morton index is shown.*

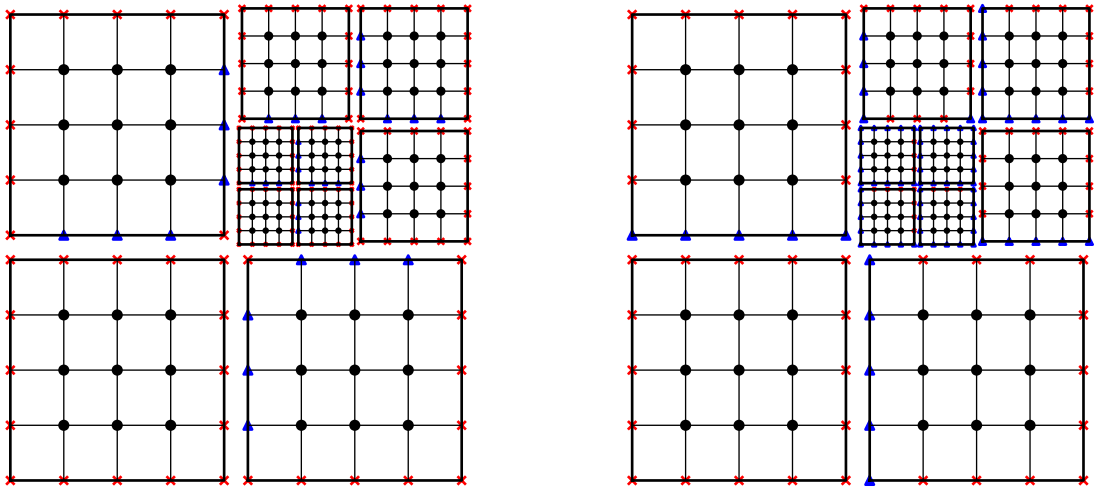


(a) *Decomposition of Ω into patches Ω_i.*

(b) *Assignment of trees $\tau_i \in (\mathbb{Z}_{\geq 0})^*$ to patches.*

(c) *Tree-based block structured non-conforming mesh.*

Figure 5.2. *Schematic description of the construction of a shallow tree mesh. The left drawing shows the coarse tessellation of the simulation domain Ω. A tree $\tau_i \in (\mathbb{Z}_{\geq 0})^*$ is assigned to each patch $\Omega_i \subset \Omega$ (middle drawing). Finally, a structured mesh is assigned to each tree leaf according to the level (right drawing).*



(a) *Geometrically non-conforming mortar subspace.* **(b)** *Conforming subspace.*

Figure 5.3. *Assignment of master and slave nodes for the mortar method (left) and the conforming subspace (right). Circles represent interior nodes, crosses identify master nodes and triangles represent slave nodes.*

we assume that the inverse mesh widths δ_ℓ^{-1} are non-decreasing powers of two so that the interface meshes induced from the non-mortar and mortar side are always nested.

We employ dual Lagrange multipliers as in the geometrically conforming case in order to obtain a sparse mortar projection. The matrix representation of the projection $\mathbf{P}$ is the same as given in Section 4.3.2. We refer to Appendix A for the discussion of our assembly strategy for the mortar projection in a geometrically conforming and geometrically non-conforming setting.

The construction of a basis of the subspace $\mathbb{Y}_{\boldsymbol{\tau}}^{\mathrm{m}}$ proceeds as in Section 4.3.5 by eliminating product-space basis functions associated to slave nodes and modifying basis functions associated to master nodes. The matrix representation of the inclusion $\mathbb{Y}_{\boldsymbol{\tau}}^{\mathrm{m}} \hookrightarrow \mathbb{X}_{\boldsymbol{\tau}}$ with respect to the basis $\boldsymbol{\pi}$ of $\mathbb{Y}_{\boldsymbol{\tau}}^{\mathrm{m}}$ and the nodal basis $\boldsymbol{\theta}$ of $\mathbb{X}_{\boldsymbol{\tau}}$ is given by

$$\mathbf{Q} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{P} \end{bmatrix} \tag{5.1}$$

when ordering the degrees of freedom of $\mathbb{Y}_{\boldsymbol{\tau}}^{\mathrm{m}}$ and $\mathbb{X}_{\boldsymbol{\tau}}$ according to interior, master and slave nodes.

5.3.2 The Subspace of Continuous Functions

As an alternative to the subspace $\mathbb{Y}_{\boldsymbol{\tau}}^{\mathrm{m}}$ obtained by enforcing weak constraints, we also consider the space

$$\mathbb{Y}_{\boldsymbol{\tau}}^{\mathrm{c}} = \mathbb{X}_{\boldsymbol{\tau}} \cap \mathcal{C}^0(\Omega)$$

of continuous functions. $\mathbb{Y}^{\mathrm{c}}_{\boldsymbol{\tau}}$ is often employed in the literature (see, for example, Burstedde et al. [36], Sampath et al. [139]), usually in combination with a 2:1 constraint on the coarse-to-fine ratio between adjacent tree leaves.

Similar to the mortar element method, the algebraic representation of the continuity constraints requires the designation of sets of *slave* and *master* nodes as depicted in Figure 5.3b. Note that the continuity condition glues together not only leaves that share a common codimension-one entity (a face if $d = 3$) but also neighboring leaves whose intersection has codimension greater than one. For a given entity (face, edge or point for $d = 3$), shared by a set of leaves $\{o_j\}$ with levels $\{\ell_j\}$, we designate nodes induced from $o_{\mathrm{argmin}_j\left(\ell_j, o_j\right)}$ as master nodes. All nodes on the entity induced from other leaves are considered slave nodes. Here, we use a lexicographical order so that $o_{\mathrm{argmin}_j\left(\ell_j, o_j\right)}$ equals the leaf with the lowest key among all leaves with the lowest level in the set $\{o_j\}$.

Let us point out that there are two important differences between the assignment of master and slave tags between the mortar element and the conforming discretization. First, in the mortar element method slave nodes only exist on the interior of codimension-one sub-entities. Second, in our implementation of the mortar method, master nodes are induced from leaves with higher levels (and therefore smaller mesh width), whereas in the conforming discretization master nodes are induced from leaves with lower keys (larger mesh width).

Using the definition of slave and master nodes as above, the continuity condition $U_{\boldsymbol{\tau}} \in \mathbb{Y}^{\mathrm{c}}_{\boldsymbol{\tau}}$ can be written as

$$(U_{\boldsymbol{\tau}})_{\dot{\alpha}} = \sum_{\beta} (U_{\boldsymbol{\tau}})_{\beta} \, \theta_{\beta}\left(\mathbf{x}_{\dot{\alpha}}\right)$$

for all slave nodes $\dot{\alpha}$ and master nodes β. Thus, the equivalent of the mortar projection is given by

$$\mathbf{P}_{\dot{\alpha}\beta} = \theta_{\beta}\left(\mathbf{x}_{\dot{\alpha}}\right) \, . \tag{5.2}$$

A basis $\boldsymbol{\pi}$ of $\mathbb{Y}^{\mathrm{c}}_{\boldsymbol{\tau}}$ can be obtained by eliminating the nodal basis functions $\theta_{\dot{\alpha}}$ associated with slave nodes and by defining

$$\pi_{\alpha} = \theta_{\alpha} + \sum_{\dot{\beta}} \mathbf{P}_{\dot{\beta}\alpha} \theta_{\dot{\beta}}$$

for master nodes. The basis functions associated to interior nodes are not modified. Using the definition (5.2) for the projection matrix $\mathbf{P}$ the matrix representation of the inclusion $\mathbb{Y}^{\mathrm{c}}_{\boldsymbol{\tau}} \hookrightarrow \mathbb{X}_{\boldsymbol{\tau}}$ is given by equation (5.1).

5.3.3 Assembly Strategy

Let us consider the assembly of the stiffness matrix corresponding to a bilinear form a. As before we denote the nodal basis of $\mathbb{X}_{\boldsymbol{\tau}}$ by $\boldsymbol{\theta}$ and the chosen basis of the subspace $\mathbb{Y}$ (which may be either a mortar or a conforming subspace) by $\boldsymbol{\pi}$. We can write

$$\pi_{\alpha} = \sum_{\beta} \mathbf{Q}_{\beta\alpha} \theta_{\beta}$$

where $\mathbf{Q}$ is the matrix representation of the inclusion $\mathbb{Y} \hookrightarrow \mathbb{X}_{\boldsymbol{\tau}}$. Hence,

$$a\left(\pi_\alpha, \pi_\beta\right) = \sum_{\gamma,\varepsilon} \mathbf{Q}_{\gamma\alpha}\, a\left(\theta_\gamma, \theta_\varepsilon\right)\, \mathbf{Q}_{\varepsilon\beta}$$

or, equivalently,

$$\mathbf{A}^{\mathbb{Y}} = \mathbf{Q}^{\mathrm{T}} \mathbf{A}^{\mathbb{X}_{\boldsymbol{\tau}}} \mathbf{Q}\,, \tag{5.3}$$

where $\mathbf{A}^{\mathbb{X}_{\boldsymbol{\tau}}}$ denotes the stiffness matrix of a on $\mathbb{X}_{\boldsymbol{\tau}}$ with respect to the basis $\boldsymbol{\theta}$. The lightweight adaptive scheme discussed in Chapter 4 uses equation (5.3) to implement the multiplication by $\mathbf{A}^{\mathbb{Y}}$ in a matrix-free fashion. Hence, only the assembly of $\mathbf{A}^{\mathbb{X}_{\boldsymbol{\tau}}}$ is required which can be done efficiently due to the known a priori structure of the stiffness matrix. However, a matrix-free setup complicates the construction of preconditioners and therefore limits the flexibility of the numerical method.

In the context of multi-level methods (e.g., algebraic multi-grid methods), equation (5.3) is used to assemble coarse grid stiffness matrix ($\mathbf{A}^{\mathbb{Y}}$) from a fine grid operator ($\mathbf{A}^{\mathbb{X}_{\boldsymbol{\tau}}}$) via the interpolation operator ($\mathbf{Q}$). This *Galerkin procedure* has a natural interpretation in terms of summed quadrature with coefficients given by the entries of $\mathbf{Q}$.

Here, we propose a third approach based on modified local-to-global mappings in the element-wise assembly of the stiffness matrix. Let us consider the construction of the stiffness matrix $\mathbf{A}^{\mathbb{Y}}$ from local stiffness matrices

$$\mathbf{A}^{\mathbb{Y}} = \sum_{E \in \mathscr{T}_{\boldsymbol{\tau}}} \left(\mathbf{S}_E^{\mathbb{Y}}\right)^{\mathrm{T}} \mathbf{A}_E^{\mathbb{Y}} \mathbf{S}_E^{\mathbb{Y}}\,, \tag{5.4}$$

where $\mathbf{A}_E^{\mathbb{Y}}$ is the local stiffness matrix with respect to the subspace basis and $\mathbf{S}_E^{\mathbb{Y}}$ the global-to-local mapping. The dimension of $\mathbf{A}_E^{\mathbb{Y}}$ depends on the number of basis functions whose support intersects E. Inserting the definition of $\boldsymbol{\pi}$ into equation (5.4) we obtain

$$\mathbf{A}^{\mathbb{Y}} = \sum_{E \in \mathscr{T}_{\boldsymbol{\tau}}} \left(\mathbf{Q}\mathbf{S}_E^{\mathbb{X}_{\boldsymbol{\tau}}}\right)^{\mathrm{T}} \mathbf{A}_E^{\mathbb{X}_{\boldsymbol{\tau}}} \left(\mathbf{Q}\mathbf{S}_E^{\mathbb{X}_{\boldsymbol{\tau}}}\right)\,. \tag{5.5}$$

with $\mathbf{A}_E^{\mathbb{X}_{\boldsymbol{\tau}}} \in \mathbb{R}^{2^d \times 2^d}$ and $\left(\mathbf{S}_E^{\mathbb{X}_{\boldsymbol{\tau}}}\right)_{i\alpha} = \delta_{\alpha\beta_i}$, if the node β_i equals the i^{th} corner of E. With the definition $\tilde{\mathbf{S}}_E = \mathbf{Q}\mathbf{S}_E^{\mathbb{X}_{\boldsymbol{\tau}}}$ we can rewrite equation (5.4) as

$$\mathbf{A}^{\mathbb{Y}} = \sum_{E \in \mathscr{T}_{\boldsymbol{\tau}}} \tilde{\mathbf{S}}_E^{\mathrm{T}} \mathbf{A}_E^{\mathbb{X}_{\boldsymbol{\tau}}} \tilde{\mathbf{S}}_E\,. \tag{5.6}$$

Equation (5.6) allows for constructing the stiffness matrix for an arbitrary subspace $\mathbb{Y}$ from the local stiffness matrix of a with respect to the standard nodal basis independently of the chosen quadrature rule. In comparison to standard finite elements methods we however have to deal with more complicated gather-scatter matrices $\tilde{\mathbf{S}}_E$. Note that $\left(\tilde{\mathbf{S}}_E\right)_{i\alpha} = \mathbf{Q}_{\beta_i\alpha}$, if the node β_i equals the i^{th} corner of E.

Similarly, we can assemble the right-hand side $\mathbf{b}^{\mathbb{Y}}$ corresponding to the bilinear form $(b,U)_{L^2(\Omega)}$ from local contributions as follows:

$$\mathbf{b}^{\mathbb{Y}} = \sum_{E \in \mathscr{T}_{\boldsymbol{\tau}}} \left(\mathbf{Q}\mathbf{S}_E^{\mathbb{X}_{\boldsymbol{\tau}}}\right)^{\mathrm{T}} \mathbf{b}_E^{\mathbb{X}_{\boldsymbol{\tau}}} = \sum_{E \in \mathscr{T}_{\boldsymbol{\tau}}} \left(\tilde{\mathbf{S}}_E\right)^{\mathrm{T}} \mathbf{b}_E^{\mathbb{X}_{\boldsymbol{\tau}}}\,.$$

5.4 Implementation and Parallelization

We implemented the proposed adaptive scheme in a new simulation code. In the following we discuss key aspects of this implementation and the parallelization scheme.

Our simulation code is written in C++ and Lua[89]. We use C++ to implement the core data structures and algorithms and provide a Lua interface to allow for extending and customizing the core library. For example, all linear solvers are implemented in C++ (or provided by a third-party library) while the non-linear Newton solver is implemented in Lua. As the individual tasks (assembling Jacobian matrices, solving linear systems and evaluating functionals) are sufficiently heavy-weight, the reduced speed of the interpreted Lua code does not impact performance. The dynamic features of the Lua language (including dynamic typing and functional programming support) simplify the implementation of the high-level logic. In previous work we gathered experience with a Python interface to a multi-scale simulation code[99]. Here, we decided to use Lua instead because of its smaller runtime which simplifies porting between supercomputers.

Our implementation targets homogeneous clusters of multicore chips (see Chapter 3) and thus uses a combination of message-passing and threading. Since our current implementation of the linear algebra classes is based on PETSC[11], which does not yet support this type of parallelization, the hybrid parallelization is (as of this writing) not fully functional and we only report results using a single compute thread per process. We therefore use the term *thread* and *processing element* interchangeably.

5.4.1 Mesh Datastructure

In our reference implementation the handling of block-structured tree-based meshes is based on three main classes: `Tessellation`, `Forest` and `Mesh`.

The fixed tessellation $\Omega = \bigcup_{i=1}^{N} \Omega_i$ underlying the adaptive mesh $\mathscr{T}_{\boldsymbol{\tau}}$ is represented by an object of type `Tessellation` which stores the same information that is used to represent uniform unstructured meshes in standard finite element codes. This includes, for example, nodal coordinates and corner indices for each patch Ω_i. Each node and element has a unique *key* in $\mathbb{Z}_{\geq 0}$ which allows to identify duplicate nodes or elements without using (error-prone) geometric comparisons. These keys are assigned in the pre-processing phase and do not change during the simulation. In our experiments we use the Morton index of the scaled integer coordinates of the element midpoint to compute the key. Elements are locally sorted according to their key.

For each patch Ω_i we store a list of patches Ω_j such that $\overline{\Omega_i} \cap \overline{\Omega_j} \neq \emptyset$, i.e., the list of all neighbors that share a sub-entity (face, edge, corner, ...) of codimension greater or equal to one. For the construction of the mortar subspace $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\tau}}$ it is sufficient to maintain a list of neighbors across codimension-one sub-entities (i.e., faces if $d = 3$), see Section 4.7. However, in order to build a conforming subspace $\mathbb{Y}^{\mathrm{c}}_{\boldsymbol{\tau}}$ we need to identify neighbors across sub-entities of codimension $1, \ldots, d$.

The elements of the tessellation are distributed over all processing elements. In general, a patch Ω_i can be stored on an arbitrary number of threads. In order to allow for coordination between

holders of a patch we store (for each patch) a list of all holders in a consistent order. In contrast to Burstedde et al.[35] we decided not to replicate the tessellation but to work with a more complicated, distributed `Tessellation` data structure since we target a very lightweight mesh data structure and want to be able to handle larger 4-dimensional tessellations (see Chapter 6) as well.

The `Tessellation` class moreover stores a list of `SideSet` objects that identify a subset of the boundary $\partial\Omega$, e.g., to impose non-homogeneous Neumann boundary conditions. A `SideSet` consists of a unique *number* and a set of pairs (key, i) where i equals the index of the codimension-one sub-entity. The pairs are sorted according to the element key allowing the assembler to iterate over patches and side-set entries at the same time. In order to implement the space-time transfer operator discussed in Section 5.4.4 or to implement periodic boundary conditions we allow for storing relations over side sets in a `SideMap` instance. Both, `SideSet` and `SideMap`, are replicated across all processes.

The `Forest` class contains a `Tessellation` object and augments it with a list of trees. In serial, we store a single tree τ_i for each patch Ω_i. As detailed in Section 5.2 we store trees as ordered vectors of 32-bit integers. By ordering the leaves according to the Morton key we can find leaves, their parents or siblings with logarithmic complexity. Since we target shallow trees, the use of 32-bit integers (instead of the more common 64-bit type[36,139]) is not a restriction. In an early version of the code we have experimented with hashed trees[163] but found the linear storage to be easier to handle since it can be trivially serialized and de-serialized for message-based exchange.

We distribute trees by means of a one-dimensional decomposition of the key space. As proposed by Burstedde et al.[38] we use the tuple (k, o) consisting of the key k of the patch and the Morton index o of the leaf to obtain a unique identifier for a tree leaf. The space of keys (ordered lexicographically) is decomposed into as many pieces as the number of threads, taking into account positive weights $w_{(k,o)} \in \mathbb{R}_{>0}$. Since both, tessellation patches and tree leaves, are ordered according to the Morton index, which defines a space-filling curve, this approach generally leads to good load balance and a good surface-to-volume ratio if the total number of leaves in the forest is sufficiently large compared to the number of threads and if the spread of weights is not too large.

Each thread stores the leaves assigned to it according to the decomposition of the key space. Moreover, we store a copy of all trees (attached to the same or a neighboring patch) adjacent to this *local tree*. In the current implementation, a thread stores a sorted list of all partitioned trees below a patch that borders an element to which a local tree is attached. This scheme could be improved by reducing the number of stored trees to the minimum needed and moreover sparsify the copies of remote trees by dropping interior leaves. Note that, since we sort tree leaves according to the (k, o) tuple, all local leaves can conceptually (i.e., not taking into account the actual data storage scheme) be traversed consecutively. Thus, storing more trees than necessary does not incur a performance penalty. On the other hand, sparsifying the copies of remote trees can potentially speedup the neighbor search. The partition of the `Tessellation` is determined by the list of trees (local or copies), i.e., a thread is designated as the holder of a patch Ω_i if and only if the thread stores a tree attached to this patch.

Finally, the `Mesh` data structure combines a `Forest` object with the local mesh widths $\boldsymbol{\delta}$. The partition of the mesh is equal to that of the `Forest`, i.e., the structured mesh on a leaf is not partitioned further. This ensures that we can take advantage of the locally structured nature of $\mathscr{T}_{\boldsymbol{\tau}}$ independently of the decomposition of the mesh.

To separate the mesh data structure from the remaining part of the code we access the mesh through the `IMesh` interface. Since the `IMesh` interface provides direct pointer-based access to the trees attached to a patch, the traversal of the mesh structure can potentially be implemented with as few as $O(N)$ virtual function calls, where N equals the number of patches. In early experiments we did not see a significant performance drop due to the introduction of the `IMesh` interface.

In order to implement an adaptive scheme using the described mesh data structure, several mesh modification functions need to be implemented. In general, these functions are written such that they do not mutate the input mesh but rather return new `Mesh` instances. Since the storage requirement for a mesh $\mathscr{T}_{\boldsymbol{\tau}}$ is sufficiently low, we can easily handle multiple meshes in memory at each point in time. By keeping the original mesh intact we simplify the control flow of the adaptive simulation since, for example, finite element spaces built on top of the input mesh need not be updated. We considered alternative approaches, e.g., using events or signals [70], but decided in favor of an explicit management of the finite element spaces in order to avoid communication intense operations being invoked as side effects. The three fundamental operations for mesh modifications are `Mesh.Adapted`, `Mesh.FlatCopy` and `Mesh.Partitioned`.

`Mesh.Adapted` takes a `Mesh` instance and a vector of marks as inputs and returns an adapted (i.e., locally coarsened or refined) `Mesh`. The marks specify whether a tree leaf should be refined (mark equals $+1$), kept (mark equals 0) or coarsened (mark equals -1). Since we are dealing with complete trees, a leaf is only replaced by the parent if all siblings are marked for coarsening as well. Moreover, we do not replace child leaves by the parent node if the child leaves are distributed across multiple trees. Hence, the parallel execution of the code can give different results than obtained by a serial run. The advantage of this approach however is that `Mesh.Adapted` can be implemented completely local without need for communication (a similar simplification is used by Burstedde et al. [38]). As in Chapter 4, mesh adaptation is based on accumulated error estimators or indicators

$$\eta_o^\Sigma = \sum_{E \in \mathscr{T}_o} \eta_E \ .$$

In contrast to the mesh data structure $\mathscr{T}_{\boldsymbol{\ell}}$ (see Section 4.2) which can be potentially subjected to arbitrary adaptations, the tree-based meshes $\mathscr{T}_{\boldsymbol{\tau}}$ implicitly store the refinement history in the trees $\{\tau_i\}_{i=1}^N$ and are therefore less flexible in the adaptation process. By focusing on shallow trees, however, we minimize this "inertia" of the adaptive meshes.

`Mesh.FlatCopy` is used to create coarse meshes that serve, for example, as a starting point for an iterative refinement procedure. For serial execution, `Mesh.FlatCopy` is functionally equivalent to repeated coarsening of the input mesh.

The third function `Mesh.Partitioned` takes an input mesh and a vector of weights and returns a repartitioned version of the mesh. As described above, the data decomposition is based on

a decomposition of the key space that approximately balances the accumulated weights across all threads. `Mesh.Partitioned` proceeds in three steps. First, the assignment of leaves to the new owner threads is computed. In a second step, a new `Tessellation` object is constructed taking into account the partition of tree leaves. This step is required since we use distributed coarse tessellations. In the third and last step, trees are exchanged between current and new owner threads.

5.4.2 Finite Element Spaces and Linear Algebra

In contrast to the approach from Chapter 4, which can be implemented without reference to a global numbering of the degrees of freedom, the implementation discussed in this chapter involves the construction of a mapping from local degrees of freedom (i.e., the index of a shape function on an element $E \in \mathscr{T}_{\boldsymbol{\tau}}$) to global degrees of freedom. Such a mapping is a prerequisite for the use of standard linear algebra data structures (e.g., a compressed row storage scheme for the system matrices).

Standard implementations of a conforming finite element discretization are based on a mapping $(E,i) \mapsto \alpha$ that assigns a global index α to the i^{th} corner of the element E. Since the number of corners (i.e., the element type) is known, this mapping can be efficiently stored as a table and used to assemble the stiffness matrix element-wise. In the context of our non-conforming discretization, in general, no single-valued function mapping local degrees of freedom to global degrees of freedom in a subspace exists because constraints may couple slave nodes to multiple master nodes. Moreover, as discussed in Section 5.3.3, a weight is associated with each pair (i,α) corresponding to the entry in the matrix representation $\mathbf{Q}$ of the inclusion. Our implementation is based on the `IVectorSpace` interface which provides a *set-valued* local-to-global mapping

$$(E,i) \mapsto \left\{ \left(\alpha, (\tilde{\mathbf{S}}_E)_{i\alpha}\right) \;\middle|\; (\tilde{\mathbf{S}}_E)_{i\alpha} \neq 0 \right\} \subset \mathbb{Z}_{\geq 1} \times \mathbb{R}_{\neq 0} \,.$$

The `ProductSpace` implementation provides a trivial implementation of `IVectorSpace` that maps the tuple (E,i) to a set $\{(\alpha,1)\}$ of cardinality one.

The ansatz spaces `MortarSubspace` and `ConformingSubspace` implement the `IVectorSpace` interface and, additionally, the interface `IVectorSubspace`. The latter defines the prototypes for the functions `Inclusion`, which is used to map a vector from the subspace into the superspace, and `CheapProjection`, which maps a vector in the superspace into the subspace by dropping values corresponding to slave nodes. In our implementation, a subspace is equivalent to a pair $(\mathbb{X}_{\boldsymbol{\tau}}, \mathbf{Q})$ where the matrix $\mathbf{Q}$ is stored in a sparse-matrix format that allows fast access to the rows (e.g., the compressed row storage (CRS) format[136]). Thus, all our ansatz spaces use the same storage scheme and only differ in their setup (i.e., the assembly of the inclusion matrix $\mathbf{Q}$). Even though it is in principle possible to build subspaces of arbitrary `IVectorSpace` instances we restrict ourselves to the construction of subspaces in a `ProductSpace` object.

In Algorithm 5.1 the assembly routine for the matrix representation of the inclusion $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\tau}} \hookrightarrow \mathbb{X}_{\boldsymbol{\tau}}$ is shown. The algorithm proceeds in two steps. First, a representation of the skeleton $\mathcal{S}$ is constructed and the dimension of the subspace is computed. At the same time, the row length of $\mathbf{Q}$ is

$\mathcal{S} \leftarrow \{\}$
for all leaves (k,o) **do** ▷ Build skeleton and compute dimension
  for all faces F of o **do**
    Find neighbor leaves $\{(k',o')\}$ across F of the same or higher level
    if $\min_{o'} \text{level}(o') > \text{level}(o)$ or $\big(\text{level}(o') = \text{level}(o)$ and $(k',o') > (k,o)\big)$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\{(k',o')\},(k,o),\{F'\},F,\sigma)\}$
      Mark interior nodes on F as slaves
      Estimate row length of $\mathbf{Q}$ for slave nodes
    end if
  end for
end for
Allocate storage for $\mathbf{Q}$ and fill rows corresponding to master nodes
for all $s \in \mathcal{S}$ **do** ▷ Assemble mortar projection
  Assemble $\mathbf{P}$ on the slave side of s
  Enter $\mathbf{P}$ into $\mathbf{Q}$ using product space column indices
end for
Map column indices from the superspace into the subspace ▷ Via hash map

Algorithm 5.1. *Assembly of the matrix* $\mathbf{Q}$ *mapping the mortar element space* $\mathbb{Y}^{\mathrm{m}}_{\boldsymbol{\tau}}$ *into the product space* $\mathbb{X}_{\boldsymbol{\tau}}$.

estimated. The construction of the skeleton requires the search for neighbor leaves. Within a tree τ_i this is done by computing the key of the neighbor (using its coordinates) and a binary search. If the neighbor lies outside of the local coordinate system of the patch, we transform coordinates into the local coordinate system of the neighbor patch. Since we attach multiple trees to each patch (see Section 5.4.1) we search all trees in order. In general, neighboring patches can be oriented differently. Therefore we need to store the corner permutation $\sigma \in \mathbb{S}_{2^{d-1}}$ that is used to translate coordinates between coordinate systems on the shared interface of the two patches.

While the construction of the inclusion matrix for the mortar element space can be computationally expensive due to the required evaluation of surface integrals over curved element faces, it is algorithmically relatively simple since only nodes on the interior of the codimension-one subentities of leaves are coupled. The construction of the inclusion matrix for the conforming ansatz space $\mathbb{Y}^{\mathrm{c}}_{\boldsymbol{\tau}}$ on the other hand requires little floating point intense calculations (since the entries of $\mathbf{Q}$ are given by baryocentric coordinate values) but is complicated by the structure of the constraints.

Algorithm 5.2 shows the steps used to assemble the inclusion matrix $\mathbf{Q}$ for the conforming ansatz space. In contrast to Algorithm 5.1 this algorithm requires neighbor search across all subentities (of which there are 2, 4, 12 and 32 for $d = 1,\ldots,4$, respectively). This is necessary to ensure

$\mathcal{S} \leftarrow \{\}$
for all leaves (k,o) **do** ▷ Build skeleton and compute dimension
    for all sub-entities e of codimension $\in \{1,\dots,d\}$ **do**
        Find neighbor leaves $\{(k',o')\}$ across e of the same or lower level
    end for
    for all sub-entities e of codimension $\in \{1,\dots,d\}$ **do**
        Find the smallest neighbor leaf (k',o') according to the lexicographical ordering of tuples $(\text{level}(o),(k,o))$ across all sub-entities of e
        if $(\text{level}(o'),k',o') < (\text{level}(o),k,o)$ **then**
            $\mathcal{S} \leftarrow \mathcal{S} \cup \{((k',o'),(k,o),e',e,\sigma)\}$
            Mark mesh nodes on F as slaves
            Estimate row length of $\mathbf{Q}$ for slave nodes
        end if
    end for
end for
Allocate storage for $\tilde{\mathbf{Q}}$ and fill rows corresponding to master nodes
for all $s \in \mathcal{S}$ **do**
    Assemble $\mathbf{P}$ on the slave side of s
    Enter $\mathbf{P}$ into $\tilde{\mathbf{Q}}$ using product space column indices
end for
$\mathbf{Q} \leftarrow \tilde{\mathbf{Q}}$ by resolving dependency chains ▷ Involves communication
Map column indices from the superspace into the subspace ▷ Via hash map

Algorithm 5.2. *Assembly of the matrix* $\mathbf{Q}$ *mapping the conforming ansatz space* $\mathbb{Y}^{\mathrm{c}}_{\boldsymbol{\tau}}$ *into the product space* $\mathbb{X}_{\boldsymbol{\tau}}$.

that we consistently assign the slave/master tag to mesh nodes. However, while we can locally decide if a node is a slave node, we cannot reliably detect master nodes since nodes on neighboring tree leaves might depend on other nodes across a third sub-entity. Thus, when assembling the matrix $\tilde{\mathbf{Q}}$ we also add entries corresponding to slave-slave couplings. This leads to the occurrence of *dependency chains* (a path in the connectivity graph of the temporary matrix $\tilde{\mathbf{Q}}$ that connects a slave node, via other slave nodes, to a master node). These dependency chains need to be iteratively resolved to obtain the matrix $\mathbf{Q}$. Since dependency chains can cross processor boundaries, this process requires unstructured communication and repeated updates of the hash maps used to map product space indices to the corresponding subspace indices.

In contrast to the handling of hanging nodes in the literature (see, for example, Burstedde et al.[35,36]) our implementation is more complicated and potentially slower since we do not assume balanced trees. Moreover, we use constraints not only for hanging nodes but also to eliminate

duplicate (geometrically coinciding) nodes.

Algorithm 5.1 and Algorithm 5.2 can be executed in parallel since the `Mesh` data structure stores copies of remote trees and thus allows for local neighbor queries. We use the `ProductSpace` instance for data exchange (such as the index mapping from the product space into the subspace). By design, the resulting decomposition of the degrees of freedom in the subspace is non-overlapping and thus resonates well with the data decomposition used by standard third-party linear algebra packages [12,77].

As mentioned earlier, we use PETSC [12] for the implementation of our linear algebra data structures so that we can take advantage of the large number of linear solvers and preconditioners implemented in, or available through, PETSC. The disadvantage of this approach is that these data structures cannot fully exploit the special meshes used in the construction of the approximation spaces.

5.4.3 Assembly Strategy

Algorithm 5.3 describes the strategy used to assemble the discrete representation of a bilinear and linear form on an `IVectorSubspace` instance based on the approach discussed in Section 5.3.3.

A challenge for the efficient implementation of Algorithm 5.3 is the variable row length of the matrices $\tilde{\mathbf{S}}_E$. We use a memory pool allocation scheme for storing the matrix column indices and entries without the performance penalty that a standard heap allocation would incur. A *pool allocator* does not keep track of individual allocations but only maintains a pointer to the beginning of the free memory region. Therefore, allocations can be implemented very efficiently as they only require the update of a single pointer. At the end of the loop body in Algorithm 5.3, the pool is collectively freed by resetting the pointer to the beginning of the pre-allocated memory. The size of the pool is chosen a priori by the user.

We restructure the sparse matrix $\tilde{\mathbf{S}}_E$ as a dense matrix of size $2^d \times L$ for some $L \geq 2^d$ by padding with zeroes and storing the corresponding column indices in a separate array of length L. Thus, the computation of the local contribution requires two dense matrix-matrix operations of `dgemm` type. In the implementation used for the experiments in Section 5.5 we use a blocking factor of 2^d for our matrix-matrix multiplication algorithm.

In order to setup the matrix structure, an upper bound for the row lengths is required prior to the computation of the matrix entries. The accurate estimation of the row lengths is a complicated task for non-conforming meshes due to the intricate shape of the support of basis functions. In the current implementation we use an upper bound of $\sum_E \left(\# \left\{ (\tilde{\mathbf{S}}_E)_{i\alpha} \neq 0 \text{ and } i \text{ corresponds to } \beta \right\} \right)^2$ for the row length of the β^{th} row. For a three-dimensional test case this bound let to an overestimation of the total number of non-zero entries by a factor of approximately three.

1: Estimate row lengths and preallocate $\mathbf{A}^{\mathbb{Y}}$
2: **for all** $E \in \mathscr{T}_{\boldsymbol{\tau}}$ **do**
3: Assemble local stiffness matrix $\mathbf{A}_E^{\mathbb{X}_{\boldsymbol{\tau}}}$ and local right-hand side $\mathbf{b}_E^{\mathbb{X}_{\boldsymbol{\tau}}}$ on E
4: Construct $\tilde{\mathbf{S}}_E$ from the rows of $\mathbf{Q}$
5: Compute $\tilde{\mathbf{S}}_E \mathbf{A}_E^{\mathbb{X}_{\boldsymbol{\tau}}} \tilde{\mathbf{S}}_E$ and $\tilde{\mathbf{S}}_E \mathbf{b}_E^{\mathbb{X}_{\boldsymbol{\tau}}}$ ▷ Matrix-matrix and matrix-vector multiplication
6: Add results to the global matrix $\mathbf{A}^{\mathbb{Y}}$ and the vector $\mathbf{b}^{\mathbb{Y}}$ ▷ Stash remote entries
7: **end for**
8: Finish assembly ▷ Exchange stashed entries

Algorithm 5.3. *Assembly of the stiffness matrix* $\mathbf{A}^{\mathbb{Y}}$ *and right-hand side* $\mathbf{b}^{\mathbb{Y}}$.

5.4.4 Transfer Operators

The experiments in Chapter 4 showed that a local transfer operator $\mathbb{X}_{\boldsymbol{\tau}(t)} \to \mathbb{X}_{\boldsymbol{\tau}(t')}$ followed by an approximate projection onto the subspace is a good alternative to a more expensive L^2-projection between subspaces. Therefore we only implemented the local transfer operator between product spaces. As defined in Section 4.5.2 this transfer operator uses local interpolation and projection on each leaf.

Our implementation is capable of transferring data between vector spaces on two arbitrary meshes as long as the underlying tessellations are identical. In particular, we do not require the decomposition of the two meshes to be identical. To this end, each thread stores the range of patch indices $\{i_{\text{low}}, \dots, i_{\text{hi}}\}$ for all threads allowing communication tasks to be set up based on the intersection between these ranges. In a next step, trees are exchanged between threads. Finally, overlapped with local computation, the vector data is transferred.

While this approach leads to a high communication volume we found it to be important not to restrict the transfer to meshes with compatible distribution since this would severely limit the flexibility of the adaptive refinement procedure. For example, in the lightweight adaptive scheme from Chapter 4 we need to store the saved dynamic variables $V, \mathbf{s}$ on the same mesh used for propagation. Thus, our choices of the initial mesh for the propagation is restricted because a very coarse mesh would lead to a large approximation error for the initial conditions and thus destroy the accuracy of the method altogether. The possibility to transfer vectors between incompatible meshes allows us to store the initial values on a separate mesh and thus aggressively coarsen the mesh used for the propagation.

In Chapter 6 a second transfer operator is required. Therein we consider domains Q of the form $\Omega \times (0,T)$, $\Omega \subset \mathbb{R}^{d-1}$, and define an operator $\mathbf{T}$ between the trace spaces on $\Omega \times \{T\}$ and $\Omega \times \{0\}$ that maps the trace $\hat{U}(\mathbf{x}) = \hat{U}(\mathbf{x}, T)$ on the upper boundary to the identical trace on the lower boundary.

To implement this operator during the mesh generation process we create a mapping from the

patch faces on the lower boundary of Q to the corresponding patch faces on the upper boundary of Q. This relation is stored as a `SideMap` instance in the `Tessellation` object. Using this relation, the transfer operator can be implemented as a local transfer operator between the trace spaces on the $(d-1)$-dimensional meshes induced on the upper and lower side of Q. Note that this transfer operator requires communication due to the d-dimensional space-filling curve data decomposition we employ.

5.5 Results

The following tests have been performed on the Cray XE6 "Monte Rosa" at the Swiss National Supercomputing Centre, featuring dual-socket nodes with AMD Interlagos CPUs, 32 GiB main memory per node and a Gemini interconnect. To avoid a negative impact of the shared floating point units in the Bulldozer microarchitecture, in all experiments we placed only one process per Bulldozer module.

Our code is compiled with optimizations using the gcc-4.7.2 compiler. In contrast to the experiments presented in Chapter 4, we used a self-compiled version of the development branch of PETSC instead of the system installation provided by Cray.

5.5.1 Small-Scale Problem

In this section we discuss the solution of the model problem from Section 4.8.2 by means of a forest of shallow trees. The monodomain equation was solved on the domain $\Omega = (0,1)^2 \times (0, \frac{1}{16})$ with fibers oriented in the xy plane with an angle of 45°. A current $I_{\text{app}} = 250\,\mu\text{A/cm}^2$ was applied in the center of the domain. The coarse tessellation consisted of $16 \times 16 \times 1$ hexahedra. In the line with the choice from Section 4.8.2 we limited the tree depth to three levels and set $(\delta_\ell)_{\ell=1}^3 = (1/4, 1/4, 1/4)$. Note that this corresponds to the choice of the mesh width from Section 4.8.2, since the diameter of tree leaves is halved in each refinement step. We employed a conjugate gradient solver with ILU preconditioner.

In contrast to the residual-based error estimator employed in Section 4.8.2 we used a gradient error indicator

$$\left(\eta_o^\Sigma\right)^2 = \int_t^{t+\tau L_{\text{lap}}} (\boldsymbol{\nabla} V_{\boldsymbol{\tau}}, \boldsymbol{\nabla} V_{\boldsymbol{\tau}})_{L^2(o)}^2 \, \mathrm{d}s$$

on the time window $(t, t+\tau L_{\text{lap}})$ here. The temporal integral was approximated by a summed trapezoidal rule and the spatial integral was evaluated via a summed midpoint rule. At the end of each lap, leaves with accumulated error indicators $\eta_o^\Sigma \geq \frac{1}{2} \max_{o'} \eta_{o'}^\Sigma$ were marked for refinement. The mesh adaptation procedure was stopped if no leaf was marked for refinement or if the sum of all error indicators was less than 1.

Before integrating over a new time window, the mesh was coarsened by cropping all trees by one level. We implemented this more aggressive coarsening (compared to the approach from Chapter 4) without reducing the accuracy by keeping the initial conditions (i.e., the final solution from the previous lap) on a separate mesh.

In Figures 5.5–5.7 the measured results are presented in the same form as in Section 4.8.2 (cf. Figures 4.7–4.9). We compare our adaptive solution method to the same structured grid solution method used in Section 4.8.2. The timings for the adaptive method include time spent in I/O.

In comparison to the lightweight adaptive approach from Chapter 4, the use of shallow trees reduces the time per lap by up to 7.58× and on average by 2.76×. Even though the adaptive code is up to 3.4× slower per lap than the structured code, on average the adaptive code is 1.37× faster per lap during the first 20 ms. The break even point, where the accumulated time of the adaptive code is lower than that of the structured code, is reached at $t = 32.5$ ms, i.e., during the repolarization phase.

Only during the simulation time interval $(19\text{ ms}, 21\text{ ms})$, consisting of four laps, a slowdown by a factor 1.5–2.75 with respect to the lightweight adaptive scheme is measured. At this time the depolarization front leaves the computational domain and our marking strategy triggers mesh refinement towards the corners of the domain. At the same time, the number of passes and the accumulated number of iterations in the linear solver (see Figure 5.6) increases. Note that, even with the same error estimator, we expect different behavior from the same marking strategy due to the different spectrum of the error indicators $\{\eta_i^\Sigma\}$ and $\{\eta_o^\Sigma\}$ for the lightweight adaptive and shallow trees approaches.

A comparison of Figures 5.7 and Figure 4.9 shows that the use of shallow trees reduces the number of degrees of freedom required for the integration of the monodomain equation. Note that in Figure 4.9 the number of mesh nodes is shown whereas we show the number of degrees of freedom in Figure 5.7. This is reasonable since the implementation in this chapter solves the discretized equations directly in the mortar element subspace. In Figure 5.4 the adaptive meshes obtained via the shallow tree approach (left) and lightweight adaptive approach (right) are shown together with the contours of the discrete solution.

In Figure 5.8 the distribution of the execution time over several components in the implicit-explicit integration scheme is shown. The solution of the linear system and the computation of the ionic current together with the integration of the state variables using the Rush-Larsen scheme are the two most costly operations. The construction of the mortar space (using a second-order Gauss-Legendre quadrature formula for the surface integrals) and the computation of the error indicators are comparably inexpensive. The assembly of mass and stiffness matrices requires a significant percentage of the computing time. In these experiments we use a variant of the algorithm discussed in Section 5.4.3 which is tailored for the considered scalar problem, avoids any virtual function calls, and reduces the computations of the Jacobian of the element transfer matrix to a minimum by reusing the same matrices for all elements on the same tree leaf. Further performance analysis of the assembly routine shows that the sparse matrix insertion of entries from elements adjacent to a slave face dominates the assembly time (with a share of more than 60 % for most laps) during the depolarization phase.

5.5.2 Large-Scale Problem

In this section we analyze the performance of the shallow tree approach for the large-scale problem **A** from Section 4.8.3. For the experiments described in the following we used a coarse tessellation

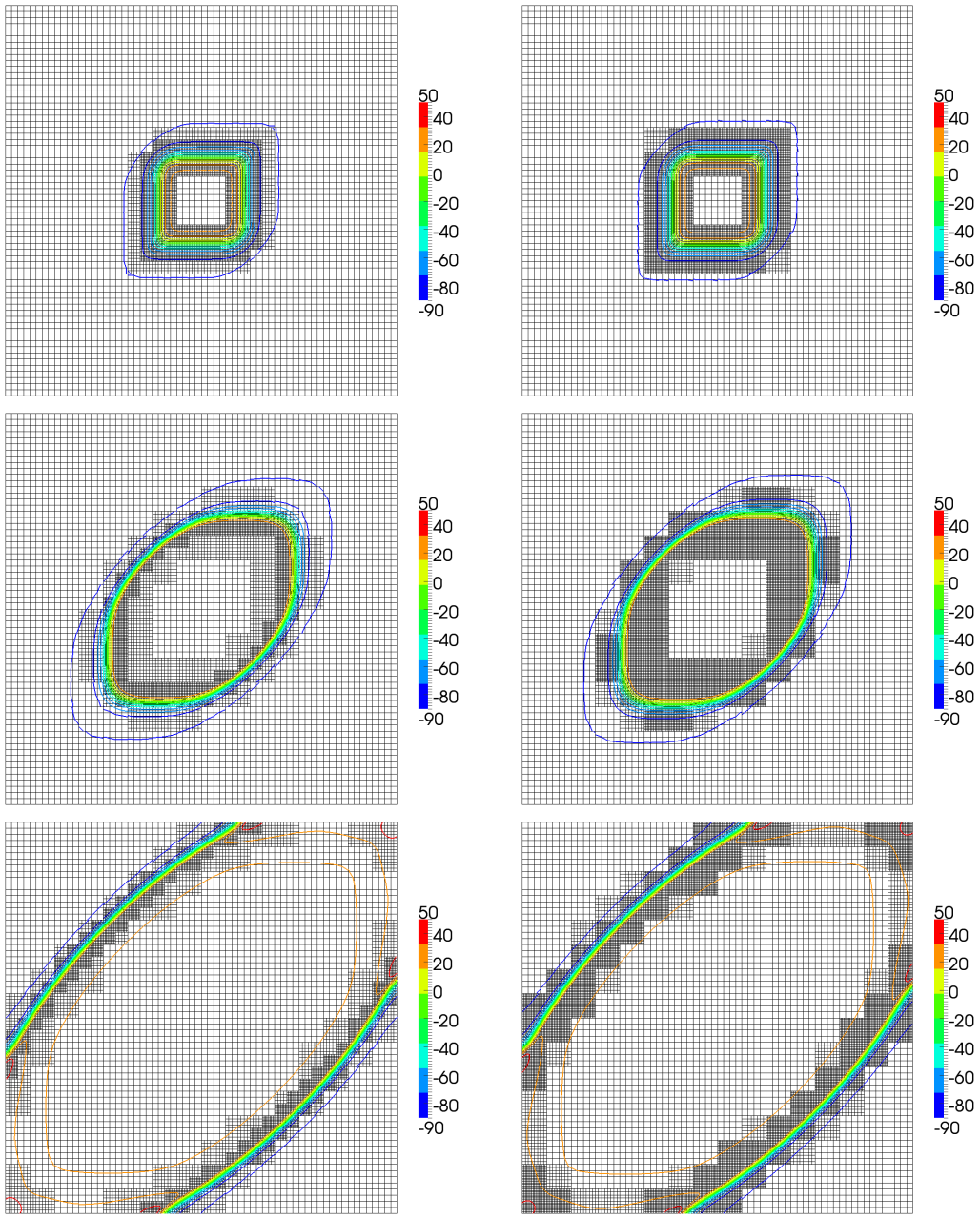


Figure 5.4. *Contours of the membrane voltage (in mV) and adaptive mesh at $t = 0.5, 1, 7.5$ ms (top to bottom) for the small-scale problem. The left plots show results obtained using our shallow tree adaptive approach. The right plots show results obtained with the lightweight adaptive approach (see Section 4.8.2).*

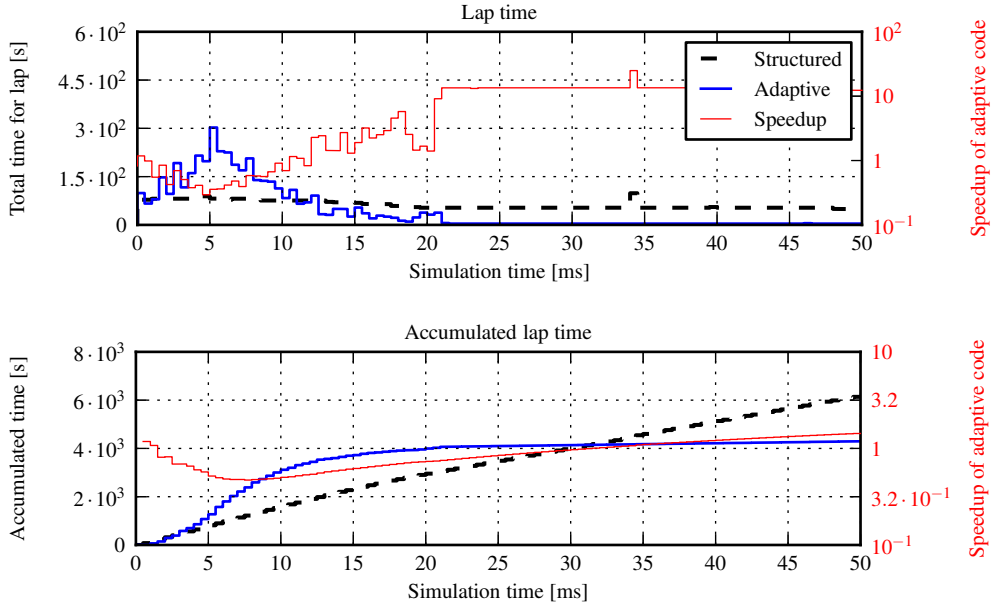


Figure 5.5. *Measured execution times. The upper graph shows the walltime for the execution of a lap of 20 time steps. Note that in the adaptive code each lap is repeated up to four times (cf. Figure 5.6). The lower plot shows the accumulated execution time.*

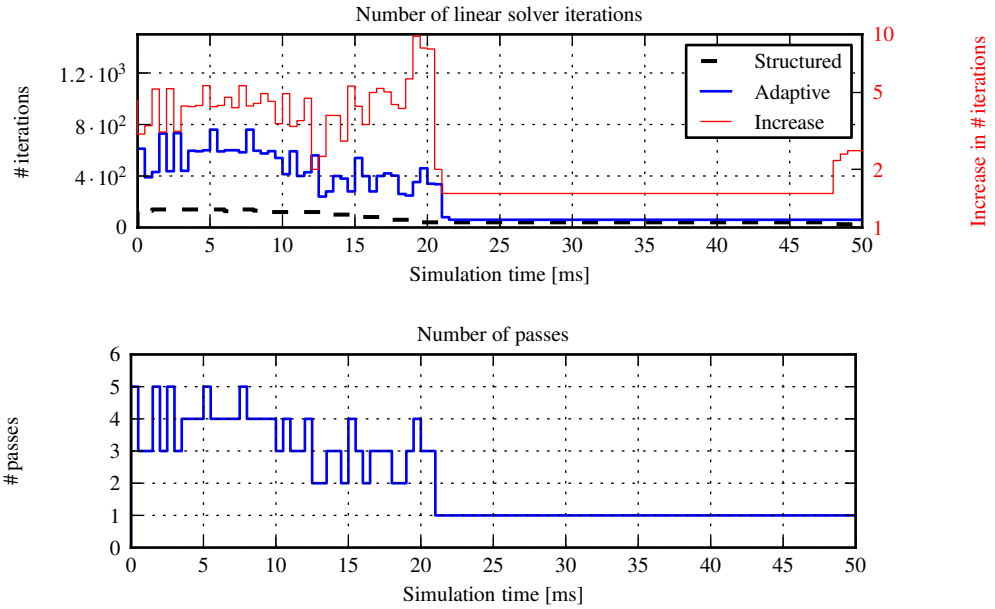


Figure 5.6. *The upper graph shows the number of linear solver iterations per lap. The lower graph shows the number of passes for the integration of a lap.*

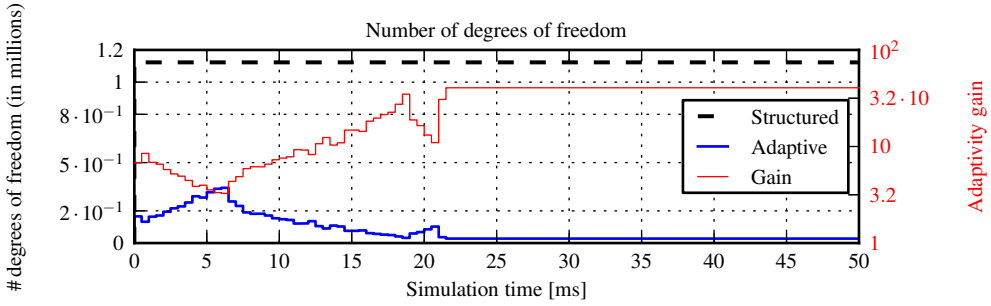


Figure 5.7. *Number of mesh nodes over time for the small-scale problem.*

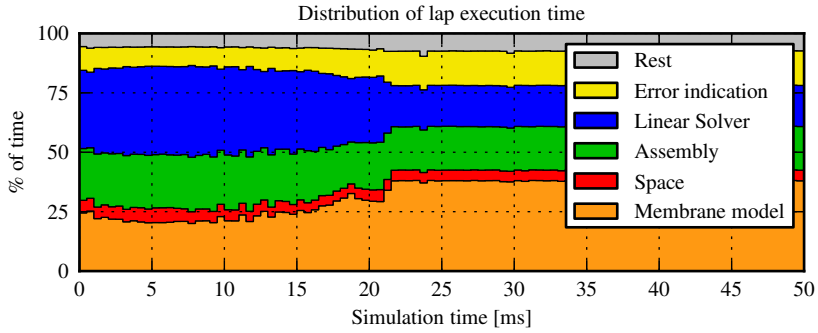


Figure 5.8. *Distribution of the execution time for the small-scale problem. The time measurements are summed over all passes over each lap.*

consisting of $4 \times 16 \times 32$ patches and $(\delta_\ell)_{\ell=1}^3 = (1/4, 1/4, 1/4)$. The simulation was run in parallel on 64 processing elements. A conjugate gradient solver with Block-Jacobi ILU preconditioner was used.

The timings for the adaptive code are reported as $64\times$ the time measured on the first processing element. Barriers have been inserted to synchronize processes prior to time measurements in order to ensure consistent results. Under the assumption of ideal scalability, these timing results correspond to the serial execution time. We compare our adaptive scheme with the optimized structured grid code from Section 4.8.3. The execution time for the adaptive code includes the time spent writing output files used for visualization and checkpointing.

In contrast to the implementation discussed in Chapter 4 we did not evaluate the exact conductivity tensor $\mathbf{G}_{\text{mono}}(\mathbf{x})$ at each quadrature point but instead used an element-wise constant approximation. This allowed us to precompute the conductivity tensors and reduce the assembly times.

The refinement strategy was based on a gradient error indicator as defined in Section 5.5.1. We marked a leaf o for refinement if $\eta_o^\Sigma \geq 0.1 \max_{o'} \eta_{o'}^\Sigma$. The adaptation procedure was stopped after the 9^{th} repetition, if no leaf was marked for refinement or if the sum of all error indicators was less than 10^2.

Before integrating over a new time window, the mesh was coarsened by cropping all trees by one level. We implemented this more aggressive coarsening (compared to the approach from Chapter 4) without reducing the accuracy by keeping the initial conditions (i.e., the final solution from the previous lap) on a separate mesh. This is possible since our implementation of the transfer operator (see Section 5.4.4) is capable of transferring data between incompatibly distributed meshes.

In Figure 5.9 and Figure 5.10 we report the execution time (per lap and accumulated), the number of linear solver iterations and the dimension of the ansatz spaces over time. In comparison to the lightweight adaptive approach, we measure a larger reduction in the number of degrees of freedom by up to 36.4% during the first 175 milliseconds of simulation time. Note that this reduction is measured for the *exact* same number of levels. The refinement procedure terminates after 3–4 passes for most laps. A similar number was measured in Section 4.8.3.

This improved adaptivity gain however is not reflected in the measured execution time which is slightly higher than the time measured in Chapter 4. First, the assembly of mass and stiffness matrices on the subspace is expensive (see below) since we cannot perform local reassembly as in Section 4.8.3. Second, the number of linear solver iterations is relatively large (with some time steps requiring up to 38 iterations) despite the lower number of blocks for the block preconditioner.

In Figure 5.12 we compare the execution time of the adaptive code with the execution time of the same code on a uniform mesh (including I/O time). The Block-Jacobi ILU preconditioned conjugate gradient solver requires 2–3 times as many iterations (11–17 iterations per solver invocation) for conforming discretization of a uniform mesh compared to the structured grid reference discretization used in Chapter 4. Even more iterations (17–28 iterations per solver invocation) are required when using a mortar discretization on a uniform mesh. Note that the mortar discretization is not equivalent to a conforming discretization even on a uniform mesh since we do not enforce continuity on the wire basket.

Compared to the conforming discretization, the adaptive code is slower during the depolarization phase and requires about 14.9% longer to finish. In comparison with the mortar discretization on the uniform mesh however, the accumulated timings (lower plot in Figure 5.12) of the adaptive code are always lower and an end-to-end speedup of 2.19 is measured.

Figure 5.13 depicts the distribution of the execution time (per lap) over several components in the implicit-explicit integration scheme. The solution of the linear system and assembly of the stiffness and mass matrix are the most expensive operations. During the depolarization phase, mesh management overhead and the transfer of the solution between different meshes does not incur a significant overhead. Note that in contrast to the results reported in Figure 4.13 we do not report communication times separately in this figure.

The computed membrane voltage V_ℓ and the adapted meshes at different steps are shown in Figure 5.11. In comparison to the results from Section 4.8.3 (see Figure 4.15), the shallow tree approach can track the depolarization front with fewer degrees of freedom due to the tree hierarchy underlying the mesh data structure.

A different approach to the construction of an initial mesh for a new time window is to start with a uniform coarse mesh, i.e., a mesh $\mathscr{T}_{\boldsymbol{\tau}}$ with all tree depths equal to zero. Since the initial conditions are stored on a separate mesh, we can use this approach without affecting the accuracy of the simulation. In comparison to the setup described above we measure a reduction of up to 7.5% in the number of degrees of freedom when starting from a coarse mesh. On average, a reduction by 1.17% is measured over the first 175 ms of simulation time. At the same time, however, the number of passes increases by 1–2 so that an increase by 13.9% in the end-to-end execution time is measured.

We can raise the adaptivity gain by increasing the maximal tree depth. To exemplify this claim we consider the large scale problem **B** from Section 4.8.3. We use mesh widths $(\delta_\ell)_{\ell=1}^4 = (1/2, 1/2, 1/4, 1/4)$. Note that the setup of from Chapter 4.8.3 corresponds to $(\delta_\ell)_{\ell=1}^3 = (1/2, 1/8, 1/8)$. The remaining setup is the same as for problem **A**. Figure 5.14 depicts the number of degrees of freedom over time. For this setup we measure a reduction by a factor of at least 13.27 which compares favorably to the results reported in Section 4.8.3. At the same time, however, in this simulation 7–8 passes are performed over each lap which is considerably larger than the number of passes measured for the lightweight scheme from Section 4.8.3.

5.5.3 Bidomain Equation

To demonstrate the flexibility of our adaptive scheme and its implementation we consider the solution of the bidomain equation in parabolic-elliptic form using a first-order splitting and an implicit-explicit Euler scheme as discussed in Section 2.3.2. We used the same setup as in Section 5.5.2 with the intra- and extra-cellular conductivity tensors

$$\begin{aligned}\mathbf{G}_\mathrm{i} &= 3.3 \cdot \mathbf{a}_\mathrm{l} \otimes \mathbf{a}_\mathrm{l} + 0.35 \cdot (\mathbf{a}_\mathrm{t} \otimes \mathbf{a}_\mathrm{t} + \mathbf{a}_\mathrm{n} \otimes \mathbf{a}_\mathrm{n}) \text{ mS/cm} , \\ \mathbf{G}_\mathrm{e} &= 2.0 \cdot \mathbf{a}_\mathrm{l} \otimes \mathbf{a}_\mathrm{l} + 1.2 \ \cdot (\mathbf{a}_\mathrm{t} \otimes \mathbf{a}_\mathrm{t} + \mathbf{a}_\mathrm{n} \otimes \mathbf{a}_\mathrm{n}) \text{ mS/cm} .\end{aligned}$$

A conjugate gradient solver with Block-Jacobi ILU preconditioner was used for the parabolic equation. The elliptic equation was solved with a Bi-CGSTAB solver preconditioned with the algebraic multi-grid BOOMERAMG[53,62]. A strong threshold value $\theta = 0.75$ was chosen. The simulation was run in parallel on 128 processing elements.

The adaptive mesh refinement was driven by a gradient indicator and the same marking strategy used in Section 5.5.2. We used a coarse mesh as the starting point for the refinement process. The error indicator only took the membrane voltage into account, i.e., the approximation error of the extra-cellular potential φ_e was not controlled by the adaptive scheme. Let us point out that this example serves only to demonstrate the feasibility of such simulations in the context of our adaptive scheme. Further verification is required to assess the accuracy of the employed refinement scheme.

In Figure 5.16 the extra-cellular potential φ_e is shown together with the adaptive meshes. For

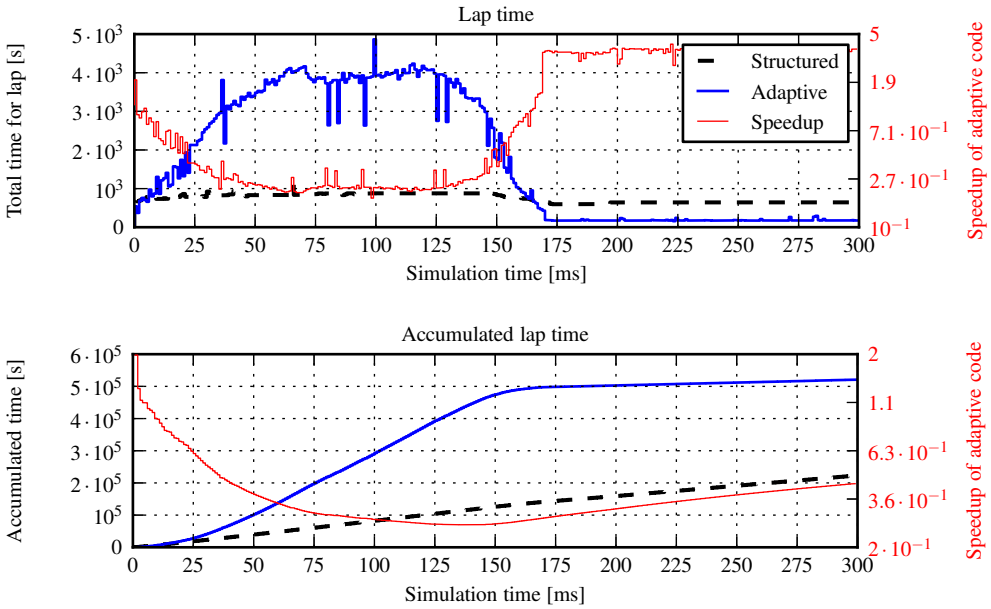


Figure 5.9. *Execution time of the adaptive code in comparison to a structured code. The upper graph shows the walltime for the execution of a lap of 20 time steps. The lower plot shows the accumulated execution time.*

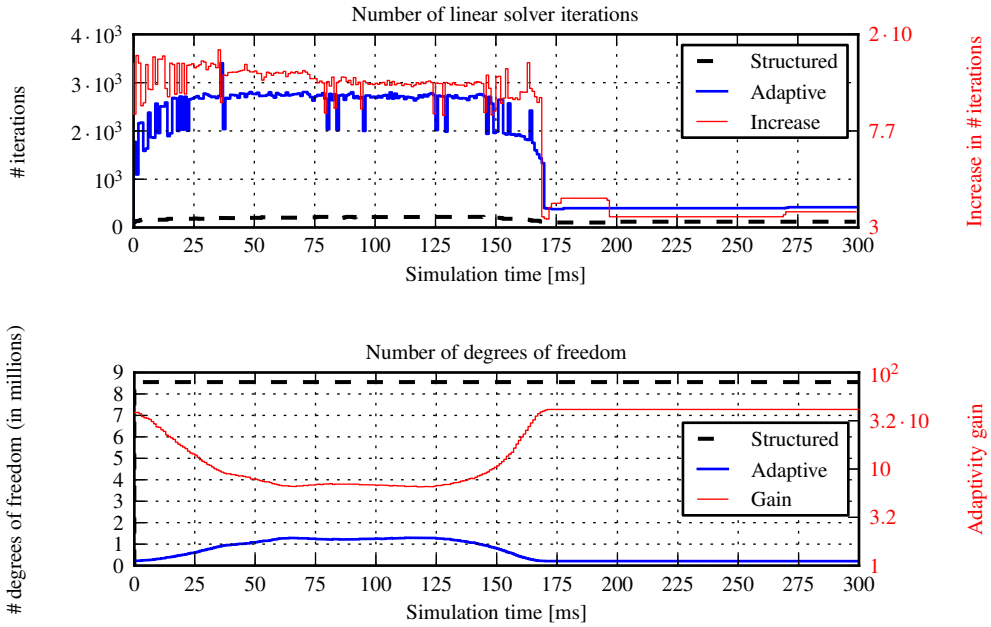


Figure 5.10. *Number of linear solver iterations (upper plot) and number of degrees of freedom (dimension of the mortar subspace) over time (lower plot).*

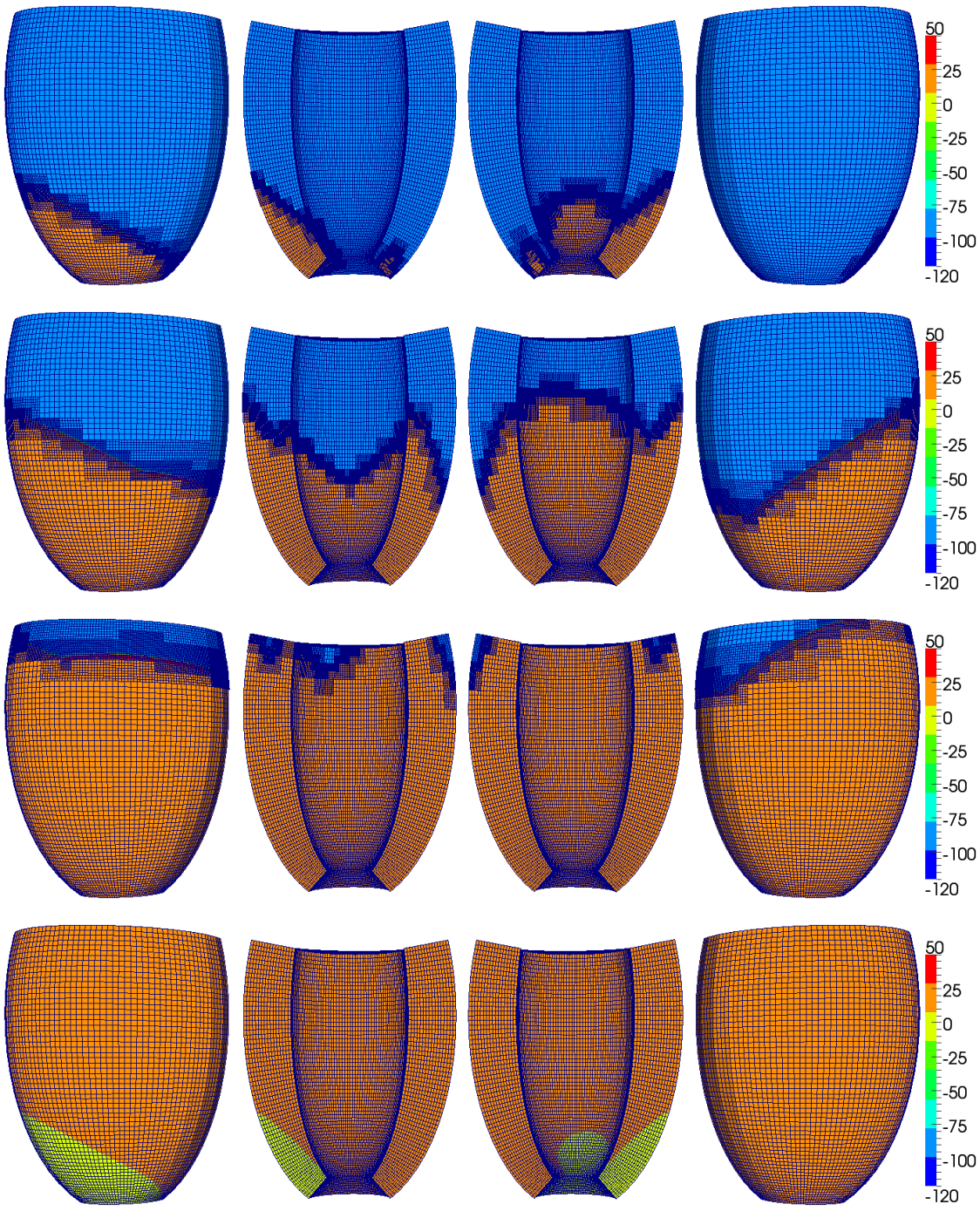


Figure 5.11. *Membrane voltage (in mV) and adaptive mesh at $t = 50, 100, 150, 200$ ms. The two plots on the right are rotated by 180° to visualize the back of the ventricle.*

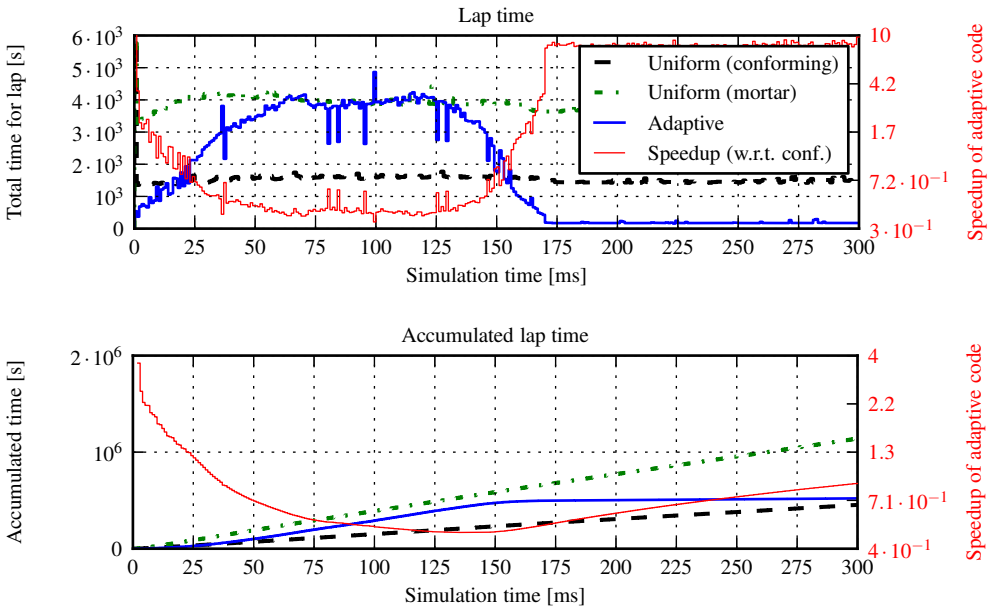


Figure 5.12. *Execution time of the adaptive code in comparison to uniform mesh methods.*

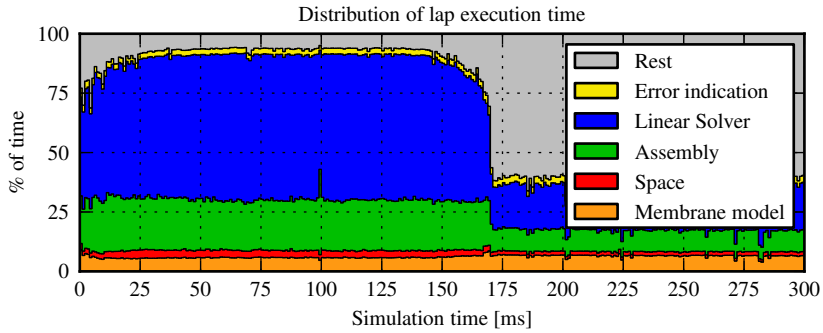


Figure 5.13. *Distribution of the execution time for the large-scale problem **A**.*

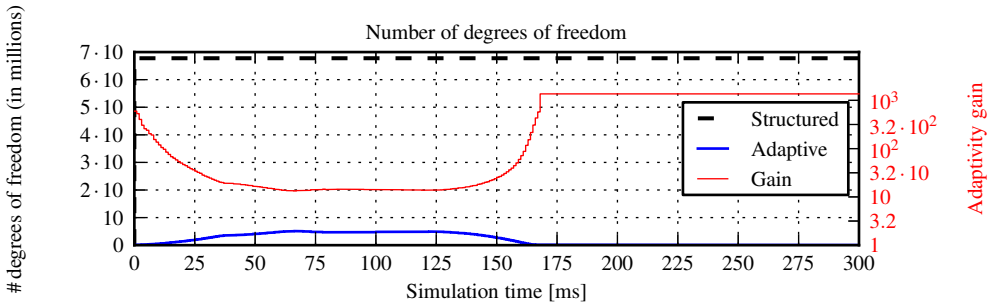


Figure 5.14. *Number of degrees of freedom for the large-scale problem **B**.*

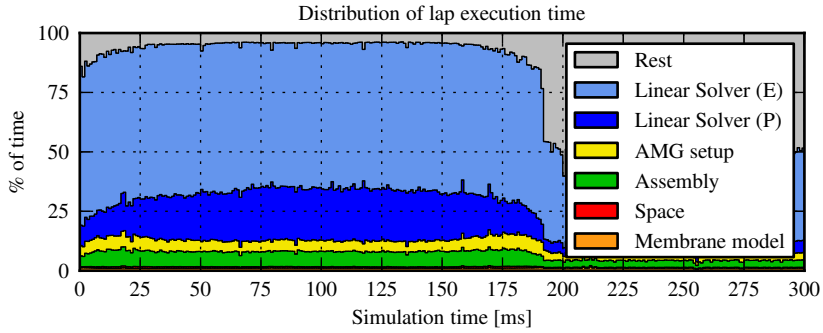


Figure 5.15. *Distribution of the execution time for the solution of the bidomain equation. The time measurements are summed over all passes over each lap.*

the visualization we have enforced

$$\int_\Omega \varphi_e(t)\,\mathrm{d}\mathbf{x} = 0 \quad \text{for all } t \in (0,T).$$

The distribution of the execution time over the computational steps of the time discretization scheme is depicted in Figure 5.15. As expected, the solution of the elliptic problem is the most compute-intense operation. The assembly of the mass and stiffness matrices and the membrane model compute phases are relatively inexpensive. As in Section 5.5.2, we find no significant overhead due to the adaptivity (i.e., mesh adaptation and transfer of solutions) during the depolarization phase.

5.5.4 Heart Model

In this section we consider the solution of the monodomain equation on a realistic heart geometry with the same model setup used by the PROPAG heart model (see Chapter 3). We use a coarse tessellation consisting of 7,974 hexahedra with an edge length of about 4 mm obtained by downsampling the PROPAG input by a factor of four in each direction. A Laplacian smoother from the MESQUITE software package was used to improve the geometry representation of the coarse tessellation. Cell types C and fiber angles were downsampled accordingly and assumed to be constant per tessellation patch. We refer to Potse et al. [127] for a detailed description of the geometric setup employed in this section.

We used a patch-wise constant conductivity tensor

$$\mathbf{G}_{\text{mono}} = 0.857 \cdot \mathbf{a}_l \otimes \mathbf{a}_l + 0.273 \cdot (\mathbf{a}_t \otimes \mathbf{a}_t + \mathbf{a}_n \otimes \mathbf{a}_n)\ \text{mS/cm}\ .$$

A current of 200 μA/cm^2 was applied for 2 ms at various times to different spatial regions according to a model of the Purkinje system (see Chapter 2). Precisely, the stimulation current was defined as

$$I_{\text{app}}(\mathbf{x},t,V) = \begin{cases} 200\ \mu\text{A/cm}^2 & \text{if } A(\mathbf{x}) \geq 0,\ t \in [A(\mathbf{x}), A(\mathbf{x}) + 2\ \text{ms}) \text{ and } V \leq -20\ \text{mV} \\ 0\ \ \mu\text{A/cm}^2 & \text{else} \end{cases}$$

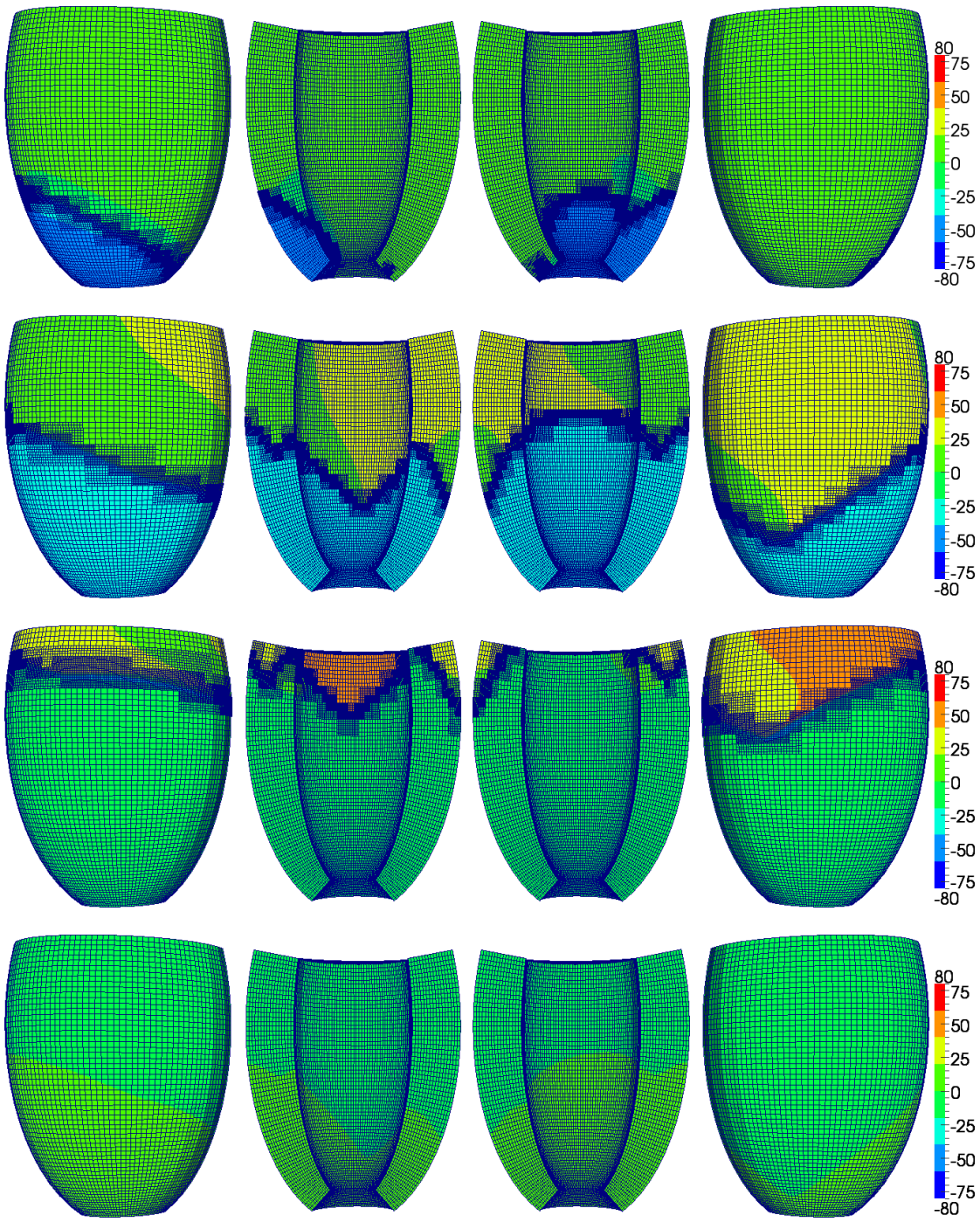


Figure 5.16. *Extra-cellular potential φ_e (in mV) and adaptive mesh at $t = 50, 100, 150, 200$ ms. The two plots on the right are rotated by 180° to visualize the back of the ventricle.*

where the activation time $A(\mathbf{x})$ equals

$$A(\mathbf{x}) = \begin{cases} C_i - 1.95 & \text{if } \mathbf{x} \in \Omega_i \text{ and } 2 \leq C_i < 98 \\ -1 & \text{else} \end{cases}.$$

We allowed adaptive meshes with up to three tree levels and set $(\delta_\ell)_{\ell=1}^3 = (1/4, 1/4, 1/4)$ corresponding to a minimal element edge length of 0.25 mm which is the resolution used for many simulations with PROPAG (problem size **M** in Section 3.4).

As in Section 5.5.2, we used a gradient error indicator and a maximum-based marking strategy. We used a coarse mesh (i.e., all tree depths equal to zero) as the initial mesh for each lap. Refinement was stopped after the 9th repetition, if no leaf was marked for refinement or if the sum of all error indicators was less than $3 \cdot 10^2$. The simulation was run on 256 processing elements.

In Figure 5.17 and Figure 5.19 the computed membrane voltage $V_{\boldsymbol{\tau}}$ during the depolarization and repolarization phase is plotted. Corresponding to the time values from Figure 5.17, the adaptive mesh is shown in Figure 5.18.

A comparative plot of the execution time of the adaptive code and of uniform mesh solution methods using either a conforming or mortar discretization is shown in Figure 5.20. The measured timings include time spent writing output files for visualization and checkpointing. As before we scale the execution time to a single processing element. In these plots the lap execution time for the mortar discretization for $t \geq 397$ is set constant since the simulation did not finish in the allocated time frame.

Similar to the results reported in Section 5.5.2 the adaptive scheme is not competitive to the conforming discretization during the depolarization time. Due to the long repolarization phase, an end-to-end speedup of 1.62 is measured. In comparison to the mortar discretization however, the adaptive code is faster over the first 100 milliseconds of simulation time and is faster by a factor of 2.99 end-to-end.

We analyzed the strong scalability of the reference implementation discussed in Section 5.4. We measured execution times (excluding I/O) with the same setup as presented above. Figure 5.21a depicts the normalized execution time of the adaptive code. As one expects, the parallel efficiency depends on the simulation time. In comparison to the lightweight adaptive approach from Chapter 4 the adaptive scheme based on shallow trees can scale to higher core counts due to the larger number of leaves available for load balancing.

Since the mesh adaptation procedure starts from a coarse initial mesh (corresponding to a 805,779 dimensional mortar space) we cannot expect ideal scalability for large core counts. In particular for the first considered lap (lap 20), the linear solver does not scale well up to 4,096 cores due to the problem size. For this lap we measure a parallel efficiency of 41.3% for the linear solver when increasing the core count from 64 to 4,096. Figure 5.21b shows the distribution of the execution time for the 20th lap. This representation of the measured data shows that the relative execution time of the assembly of mass and stiffness matrices as well as the mesh partitioning (not

separately listed) grows during scale-out. The sub-optimal scaling of the assembly routine is most likely caused by the increase of remotely computed matrix entries that are stashed and exchanged at the end of the assembly algorithm (see Algorithm 5.3).

As mentioned in Section 5.4 several optimizations are possible to our current implementation that potentially improve the scalability of the partitioning algorithm and the construction of the approximation spaces.

In comparison to the PROPAG heart model, the presented adaptive heart model has two major shortcomings. First, the use of patch-wise constant cell types affects the accuracy of the model since it artificially increases the support of the applied current. Similarly, the patch-wise constant approximation of the fiber angles could affect the accuracy even though we expect a lower impact due to the smoothness of the angles. Second, the use of a downsampled (smoothed) voxel geometry leads to a reduced accuracy of the geometry approximation.

The first shortcoming can be addressed by storing the original cell types separately and use a piece-wise constant interpolation or projection operator to compute the effective cell type on an adaptive mesh. For the considered heart geometry this requires $4^3 \cdot N = 64 \cdot N$ bytes (instead of N bytes) of main memory.

In order to improve the geometry approximation one has to take advantage of the flexibility of the employed unstructured coarse tessellation and directly create hexahedral meshes from the surface representations obtained by segmenting medical imaging data. Note however that the creation of high-quality hexahedral meshes for arbitrary geometries is still a complicated and labor intense task.

Despite the above mentioned model limitations, our results show the viability of shallow tree-based adaptivity for use in large-scale heart models. We expect that in the coming years, we will be able to deploy this technology within, for example, the patient-specific pipeline operated by the Cardiocentro Ticino and the Institute of Computational Science together with several project partners.

5.6 Discussion

We have discussed the extension of the lightweight adaptive scheme from Chapter 4 by means of a forest of trees approach. The goal of this work was to increase the flexibility of the method twofold. On the one hand, we aimed at a more fine grained control over the refinement process to increase the adaptivity gain and improve the parallel scalability. On the other hand, we targeted an adaptive scheme that could be applied to a wider variety of partial differential equations and in particular can be employed with different solution and preconditioning schemes. We combine the advantages of the lightweight adaptive scheme with the forest of trees approach by Burstedde et al.[35,38]. In contrast to other tree-based methods we focus on shallow trees with only few levels but structured tensor meshes attached to the leaves. The maximal tree depth and the width of the structured meshes provide control over the "granularity" of the adaptive scheme.

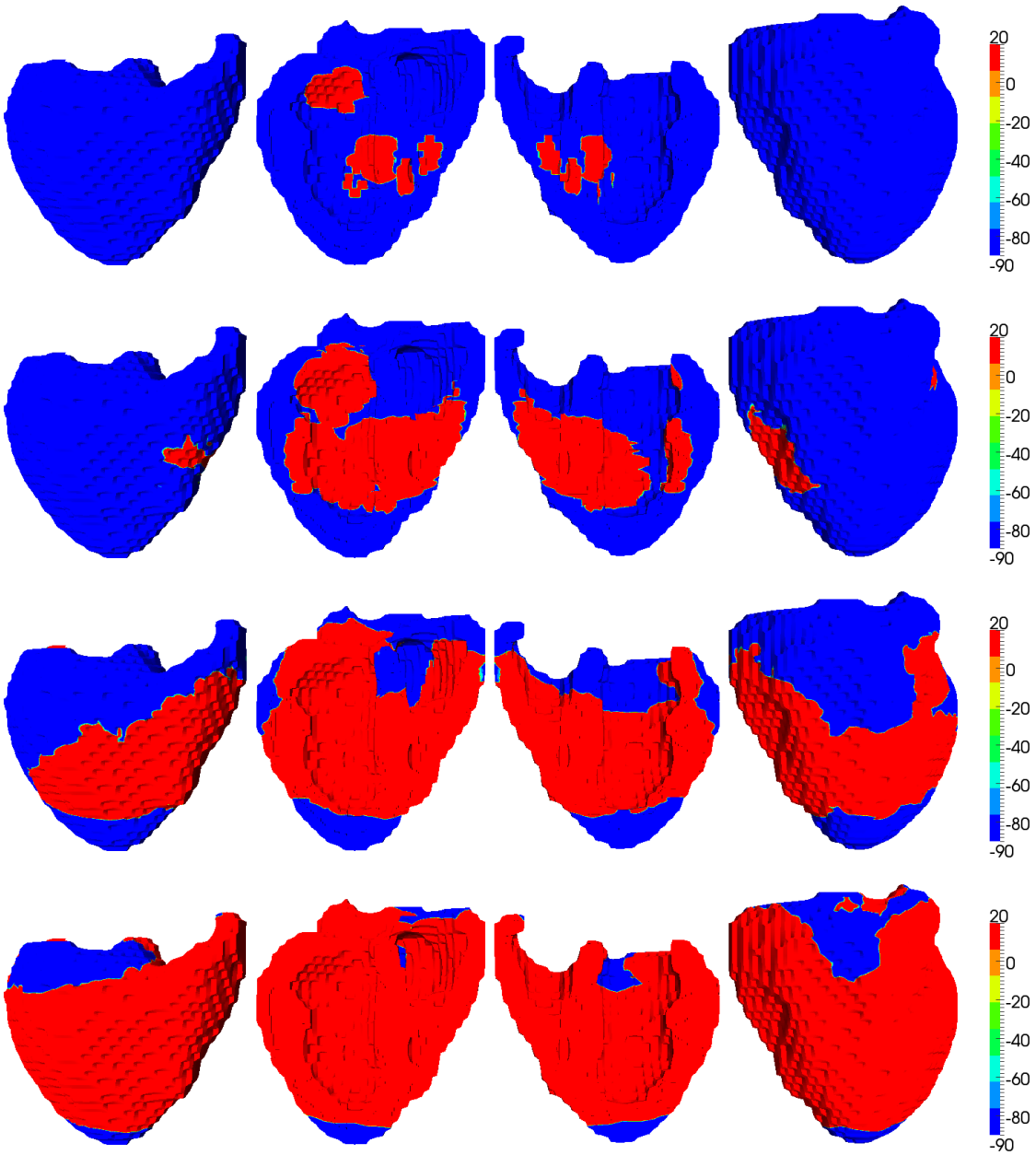


Figure 5.17. *Membrane voltage (in mV) during the depolarization phase at times $t = 15, 30, 50, 75$ ms. The two plots on the right are rotated by 180° to visualize the back of the heart. The color bar limits are set to -90 mV and 20 mV.*

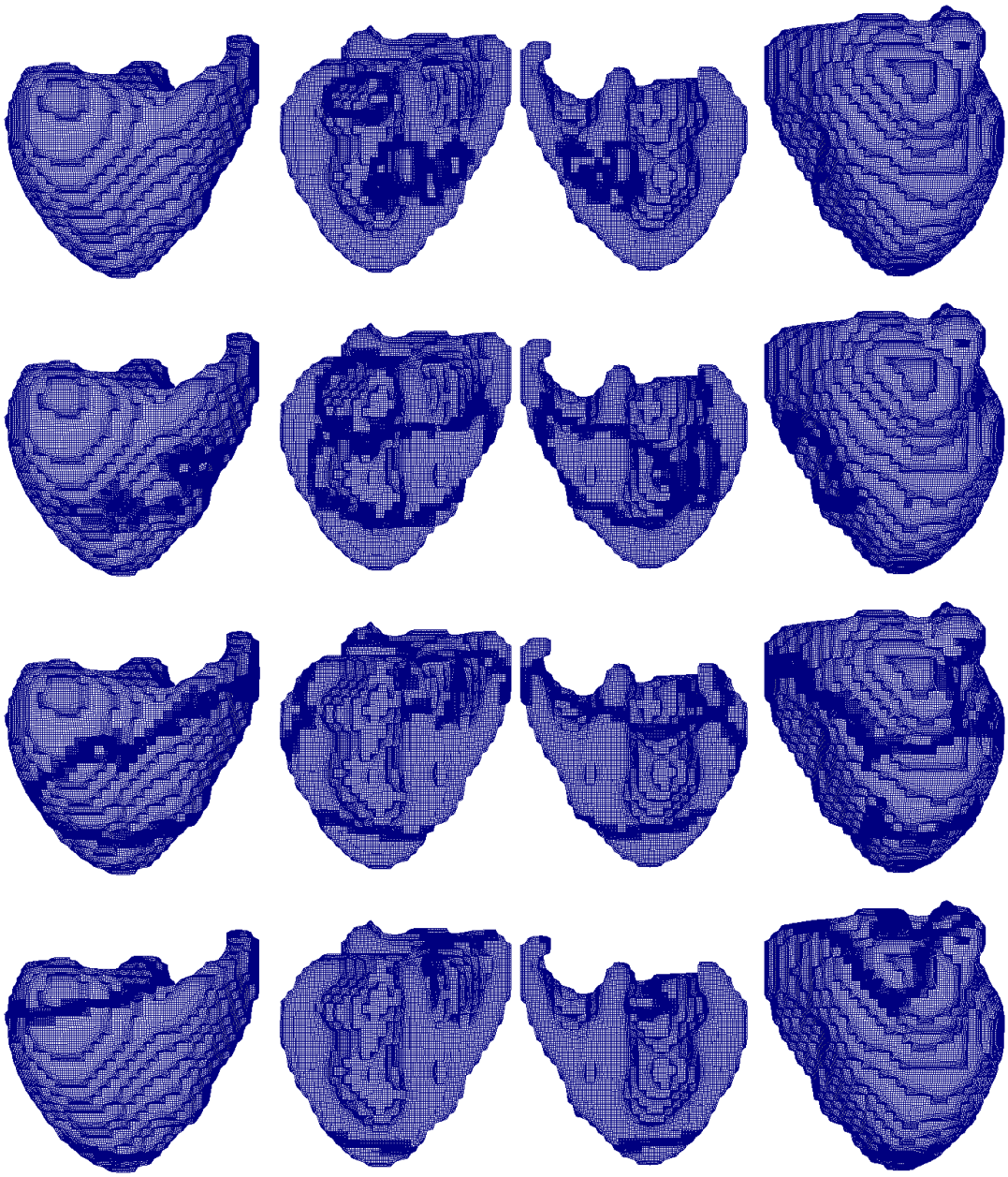


Figure 5.18. *Adaptive meshes at times $t = 15, 30, 50, 75$ ms (cf. Figure 5.17). The two plots on the right are rotated by 180° to visualize the back of the heart.*

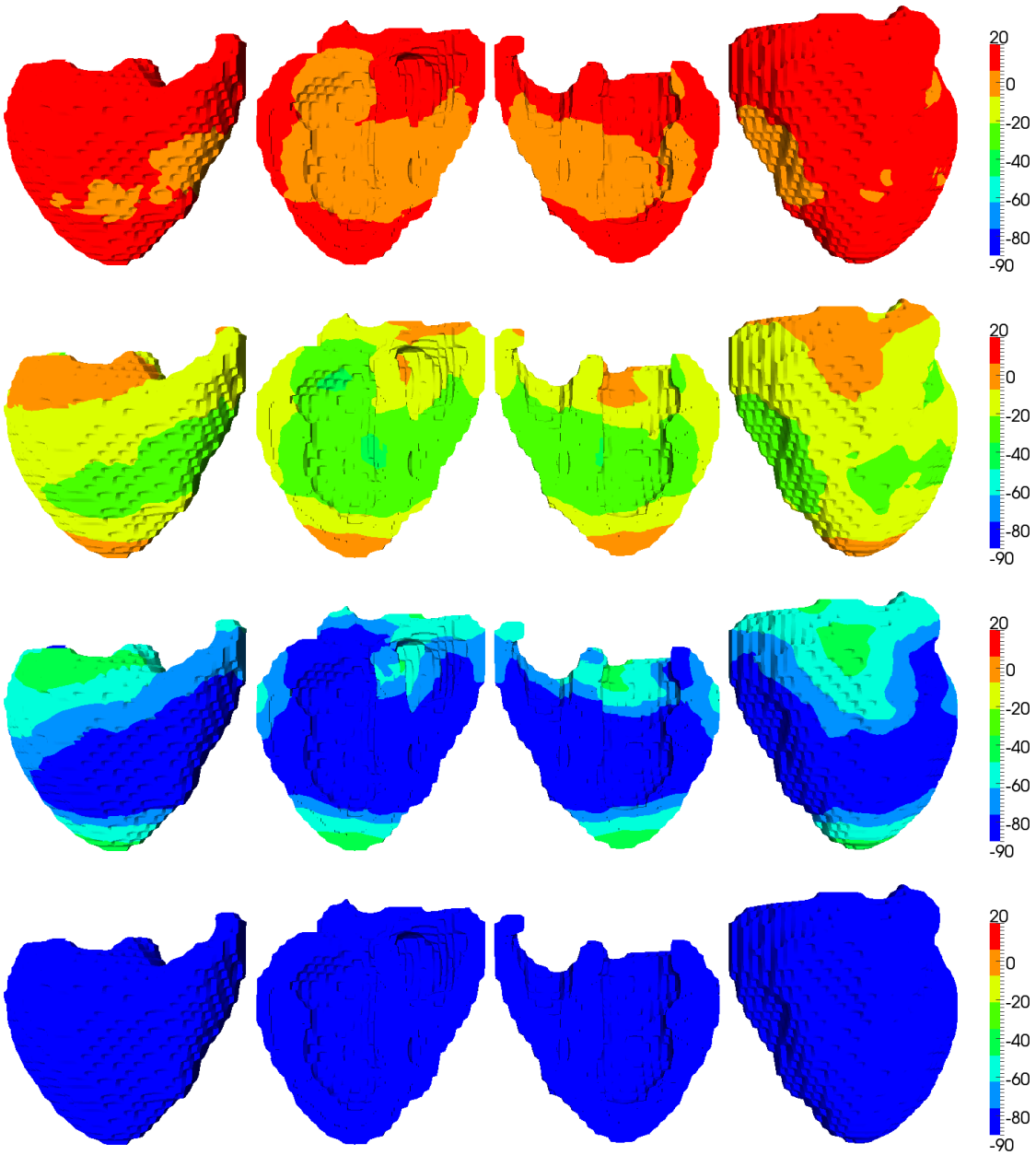


Figure 5.19. *Membrane voltage (in mV) during the repolarization phase at times $t = 200, 300, 400, 500$ ms. The two plots on the right are rotated by 180° to visualize the back of the heart. The color bar limits are set to -90 mV and 20 mV.*

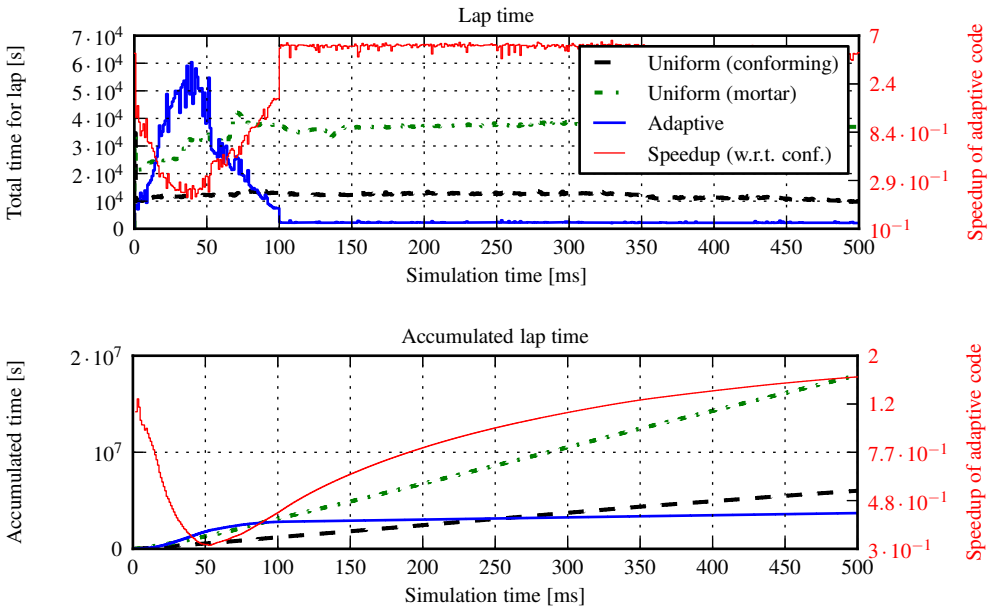
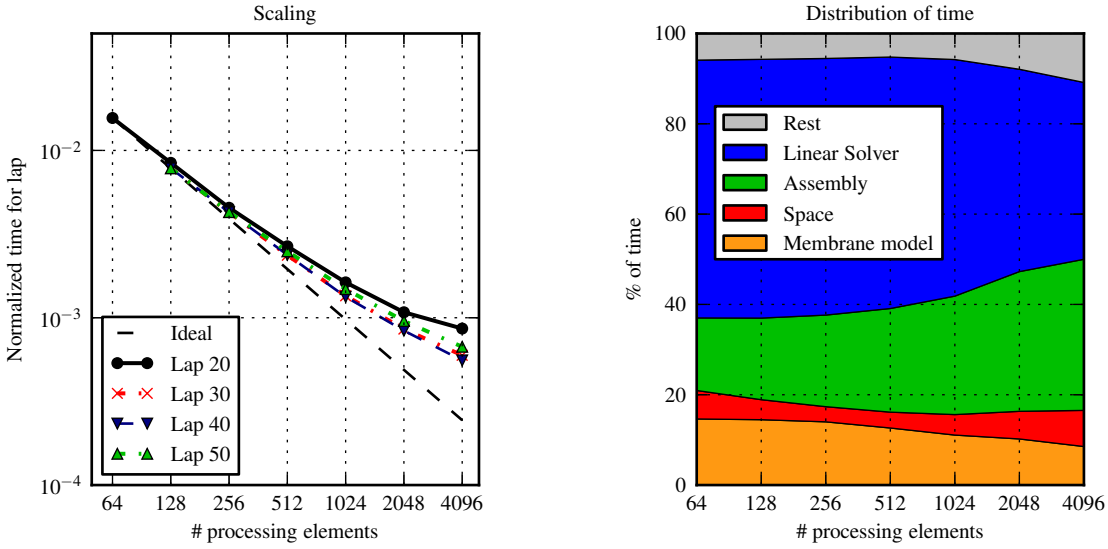


Figure 5.20. *Execution time of the adaptive code in comparison to uniform mesh solution methods. The upper graph shows the walltime for the execution of a lap of 20 time steps. The lower plot shows the accumulated execution time.*



(a) *Normalized execution times of the adaptive code (logarithmic scale).*

(b) *Distribution of the execution time for the 20th lap in dependence of the number of processing elements.*

Figure 5.21. *Strong scaling results.*

In Section 5.4 we discussed our reference implementation of the shallow tree approach. In this code, ansatz spaces are constructed as subspaces of the product space and can be represented in memory by means of the algebraic representation of the inclusion matrix. Our assembly strategy (see Section 5.3.3) allows us to assemble stiffness matrices on arbitrary subspaces without modification of the definition of the bilinear form. This flexibility is used extensively in the next chapter to study different discretization schemes using conforming and non-conforming ansatz spaces.

Our results reported in Section 5.5 show that the shallow tree approach does meet the design goals. First, we showed that through an appropriate choice of the maximal tree depth and the local mesh width, a significant reduction in the number of degrees of freedom compared to the lightweight adaptive scheme can be obtained. Second, we exemplified the flexibility of the method by considering the adaptive solution of the bidomain equation. In Section 5.5.4 we demonstrated the potential of our shallow tree adaptive scheme for use in realistic, large-scale heart models.

Based on the analysis of our results, we identify three directions for future research that have the biggest potential to significantly lower the execution time: The preconditioning technique, the marking strategy and the linear algebra data structures.

The results from Section 5.5.2 as well as from Section 4.8.3 clearly show that block-wise preconditioning, while very efficient on uniform meshes, is not equally well-working for non-conforming discretizations, even on uniform meshes. Therefore, domain decomposition preconditioners and geometric multi-level schemes should be investigated. Wohlmuth and Krause [173] developed a multi-grid scheme for mortar element discretizations using a modified interpolation operator. Even though the trees used in our shallow tree mesh implicitly store a hierarchy, the meshes employed in this chapter have no multi-level structure. Sundar et al. [149] discuss a possible approach for implementing a multi-grid scheme by using a sequence of meshes. A particular challenge will be the development of methods that are sufficiently robust while still fast enough to be competitive to the block preconditioners for conforming discretizations on uniform meshes.

We expect a large impact on the measured execution times (in particular when using deeper trees) by developing tailored marking strategies. In this work we use a maximum-based strategy that is well established for unstructured AMR methods. However, the use of spatially accumulated error estimators or indicators has a profound impact on the distribution of the former. Even though we have tuned the constants used in the marking strategy, it is likely that a tailored marking strategy can significantly reduce the number of repetitions required for a time window. In the numerical experiments reported in Section 4.8 and Section 5.5, the adaptation of the mesh is stopped if either the error is small enough (which only happens during the depolarization phase due to the upper bound on the level) or if no patches or leaves are marked for refinement. As we noted in Section 4.8.2, the premature termination of the adaptation process can spoil the accuracy of the method. Assuming we can provide an approximate measure for the "optimality" of the current mesh with respect to a structured mesh (in contrast to the comparison with the exact solution, which is the goal of error estimation), we can control the number of repetitions adaptively. Another potentially viable

approach is to use knowledge about the solution, such as the depolarization speed, to estimate the trajectory of the depolarization front and refine in advance to reduce the number of trial-and-error steps.

Our current implementation of the linear algebra data structures does not fully exploit the special structure of the meshes. For example, a compressed row storage scheme for the inclusion matrix $\mathbf{Q}$ allows for a simple implementation of the inclusion operation via a distributed sparse-matrix multiplication but cannot take advantage of the block structure (and, in particular, the local structure) of the projection matrix $\mathbf{P}$. In the lightweight adaptive scheme only the matrix entries of $\mathbf{P}$ are stored for each face which eliminates the need to store the unit vector rows of $\mathbf{Q}$ for interior and master nodes and does not require separate storage of the column indices. The generalization of this scheme to a geometrically non-conforming mortar discretization is however non-trivial. Similarly, our performance measurements from Section 5.5.1 indicate that a large percentage of the assembly time is required for the insertion of matrix entries computed on slave faces. Taking into account the classification of product space degrees of freedom into interior, master, and slave degrees of freedom it might be possible to improve the sparse-matrix data structures for our use case.

The focus of the techniques presented so far was on spatially adaptive methods. This work can be combined with step size control for the time discretization. However, global time step control is often inefficient for problems of interest in computational electrocardiology[166]. To address this challenge, in the next chapter we discuss an approach to local time stepping based on a space-time discretization of the governing equation. We employ a discretization scheme that allows for reusing the presented mesh data structures.

6 Adaptivity Using Space-Time Finite Elements

In the previous chapters we discussed techniques to solve (non-linear) reaction-diffusion equations on spatially adaptive meshes. In this chapter we turn our attention to temporal adaptivity as well as combined space-time adaptivity. As a matter of fact, global time step control is often inefficient for the equations we are interested in because the global time step is kept low by spatially localized but propagating features of the solution[166]. Local time stepping schemes, which utilize spatially varying time step sizes τ, are not readily compatible with implicit or semi-implicit discretizations of the considered equations. Here, we propose the use of non-conforming space-time finite element meshes to enable local time stepping in a rigorous manner and experimentally assess the feasibility of this approach.

6.1 Introduction

Local time stepping is a standard technique in structured adaptive mesh refinement codes for the solution of hyperbolic problems using explicit time integration[21]. To sketch the underlying idea, we consider the explicit Euler discretization of the ordinary differential equation

$$\dot{\mathbf{V}} = \mathbf{A}\mathbf{V} \tag{6.1}$$

obtained by discretizing a partial differential equation on a domain $\Omega = \Omega_1 \cup \Omega_2$. We can write $\mathbf{V} = [\mathbf{V}_1, \mathbf{V}_2]^{\mathrm{T}}$ (corresponding to two different domains) and

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} .$$

Assuming that the solution on Ω_2 requires a smaller time step, we can state a local time stepping scheme, which uses half the time step in the second domain, as

$$\begin{aligned} \mathbf{V}_1^{i+1} &= \mathbf{V}_1^i &&+ \tau \mathbf{A}_{11}\mathbf{V}_1^i &&+ \tau \mathbf{A}_{12}\mathbf{V}_1^i \\ \mathbf{V}_2^{i+\frac{1}{2}} &= \mathbf{V}_2^i &&+ \frac{\tau}{2}\mathbf{A}_{21}\mathbf{V}_1^i &&+ \frac{\tau}{2}\mathbf{A}_{22}\mathbf{V}_2^i \\ \mathbf{V}_2^{i+1} &= \mathbf{V}_2^{i+\frac{1}{2}} &&+ \frac{\tau}{2}\mathbf{A}_{21}\frac{1}{2}\left(\mathbf{V}_1^i + \mathbf{V}_1^{i+1}\right) &&+ \frac{\tau}{2}\mathbf{A}_{22}\mathbf{V}_2^{i+\frac{1}{2}} . \end{aligned} \tag{6.2}$$

Note that in the last computation, we use interpolated values from the first domain. This scheme can be easily generalized to higher coarse-to-fine ratios. Writing equation (6.2) in a matrix-formulation, we obtain

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ -\frac{\tau}{2}\mathbf{A}_{21} & -\left(\mathbf{1}+\frac{\tau}{4}\mathbf{A}_{22}\right) & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^{i+1} \\ \mathbf{V}_2^{i+\frac{1}{2}} \\ \mathbf{V}_2^{i+1} \end{bmatrix} = \begin{bmatrix} \mathbf{V}_1^{i} + \tau\mathbf{A}_{11}\mathbf{V}_1^{i} + \tau\mathbf{A}_{12}\mathbf{V}_2^{i} \\ \mathbf{V}_2^{i} + \frac{\tau}{2}\mathbf{A}_{21}\mathbf{V}_1^{i} + \frac{\tau}{2}\mathbf{A}_{22}\mathbf{V}_2^{i} \\ \frac{\tau}{4}\mathbf{A}_{21}\mathbf{V}_1^{i} \end{bmatrix}$$

Due to the lower-triangular structure of the matrix on the left-hand side, the method can be implemented via multiple explicit updates of $\mathbf{V}_2$. Moreover, it is not necessary to store $\mathbf{V}_2^{i+\frac{1}{2}}$ separately.

By introducing the auxiliary variable $\mathbf{V}_2^{i+\frac{1}{2}}$ it is straightforward to state the equivalent to equation (6.2) for an implicit Euler discretization of (6.1):

$$\begin{aligned} \mathbf{V}_1^{i+1} &= \mathbf{V}_1^{i} &&+ \tau\mathbf{A}_{11}\mathbf{V}_1^{i+1} && + \tau\mathbf{A}_{12}\mathbf{V}_1^{i+1} \\ \mathbf{V}_2^{i+\frac{1}{2}} &= \mathbf{V}_2^{i} &&+ \frac{\tau}{2}\mathbf{A}_{21}\frac{1}{2}\left(\mathbf{V}_1^{i} + \mathbf{V}_1^{i+1}\right) && + \frac{\tau}{2}\mathbf{A}_{22}\mathbf{V}_2^{i+\frac{1}{2}} \\ \mathbf{V}_2^{i+1} &= \mathbf{V}_2^{i+\frac{1}{2}} &&+ \frac{\tau}{2}\mathbf{A}_{21}\mathbf{V}_1^{i+1} && + \frac{\tau}{2}\mathbf{A}_{22}\mathbf{V}_2^{i+1} \ . \end{aligned} \tag{6.3}$$

Equation (6.3) can be written as a coupled linear system

$$\begin{bmatrix} \mathbf{1}-\tau\mathbf{A}_{11} & \mathbf{0} & -\tau\mathbf{A}_{12} \\ -\frac{\tau}{4}\mathbf{A}_{21} & \mathbf{1}-\frac{\tau}{2}\mathbf{A}_{22} & \mathbf{0} \\ -\frac{\tau}{2}\mathbf{A}_{21} & -\mathbf{1} & \mathbf{1}-\frac{\tau}{2}\mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^{i+1} \\ \mathbf{V}_2^{i+\frac{1}{2}} \\ \mathbf{V}_2^{i+1} \end{bmatrix} = \begin{bmatrix} \mathbf{V}_1^{i} \\ \mathbf{V}_2^{i} + \frac{\tau}{4}\mathbf{A}_{21}\mathbf{V}_1^{i} \\ \mathbf{0} \end{bmatrix} \ .$$

In contrast to the local time stepping scheme (6.3) based on an explicit Euler discretization, local time stepping in an implicit setting leads to a coupled system of equations which needs to be solved at once. Hence, local time stepping in an implicit setting naturally leads to a space-time formulation of the considered equation. Moreover, since the half-step values are only computed for the second domain, one may interpret the local time stepping as based on *non-conforming* space-time meshes.

Based on these insights, we propose to use non-conforming space-time meshes to enable local time stepping in an implicit setting. By using space-time finite elements on top of these non-conforming meshes, we have a flexible handling of different equations. In order to decouple the solution on different time slabs and control the computational cost of the method, we use a combination of a finite element method with a discontinuous Galerkin method[92].

6.2 Space-Time Discretization

In this section we consider the discretization of non-linear, scalar reaction diffusion equations of the form

$$\partial_t V = \boldsymbol{\nabla}\cdot(\mathbf{a}\boldsymbol{\nabla}V) + F(\mathbf{x}, V, \boldsymbol{\nabla}V) + b(\mathbf{x},t) \tag{6.4}$$

with $\mathbf{a} \in \mathcal{C}^1\left(\mathbb{R}^d, \mathbb{R}^{d\times d}\right)$ and $F \in \mathcal{C}^0\left(\mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d\right)$. We restrict ourselves to scalar equations solely to simplify the notation. Moreover, we assume homogeneous Neumann boundary conditions on the

boundary of the domain Ω.

Since we are targeting an implementation using the approach detailed in Chapter 5 we only discuss methods that can be implemented on meshes $\mathscr{T}_{\boldsymbol{\tau}}$. In particular we do not cover methods that build the space-time meshes on the fly (see, for example, Abedi et al.[2]).

6.2.1 Discretization with Continuous Finite Elements

Given a spatial approximation space $\mathbb{Y}^{\text{space}}$ built on Ω and a temporal approximation space $\mathbb{Y}^{\text{time}}$ on $(0,T)$ we can form a space-time approximation space

$$\mathbb{Y} = \mathbb{Y}^{\text{space}} \otimes \mathbb{Y}^{\text{time}}$$

on the space-time domain $Q = \Omega \times (0,T)$. The weak formulation of equation (6.4) reads as: Find $V \in \mathbb{Y}$ so that

$$(\partial_t V, U)_{L^2(Q)} + (\mathbf{a}\boldsymbol{\nabla} V, \boldsymbol{\nabla} U)_{L^2(Q)} - (F(\cdot, V, \boldsymbol{\nabla} V), U)_{L^2(Q)} = (b, U)_{L^2(Q)} \tag{6.5}$$

for all $U \in \mathbb{Y}$. Such a formulation has been studied by French and Peterson[65] and Anderson and Kimn[5] for the approximation of the wave equation (using an auxiliary variable to obtain a first-order system). Since this formulation requires the solution of the complete system at once solver cost may be prohibitive, in particular when considering a large end time T.

In order to break the global dependency in the system (6.5) one has to use discontinuous test functions. Aziz and Monk[6] define $\mathbb{Y}^{\text{time}}$ as the space of continuous piece-wise polynomial functions on a decomposition $\overline{(0,T)} = \bigcup_i [t_i, t_{i+1}]$ and use $\partial_t U$ (instead of U) as the test functions. Hence, the weak formulation of equation (6.4) is given by

$$(\partial_t V, \partial_t U)_{L^2(Q)} + (\mathbf{a}\boldsymbol{\nabla} V, \boldsymbol{\nabla}\partial_t U)_{L^2(Q)} - (F(\cdot, V, \boldsymbol{\nabla} V), \partial_t U)_{L^2(Q)} = (b, \partial_t U)_{L^2(Q)} \ . \tag{6.6}$$

This formulation is equivalent to a Petrov-Galerkin method with the test space equal to the image space $\partial_t \mathbb{Y}^{\text{time}}$. Since $\partial_t U$ can be discontinuous, equation (6.6) can be split into a series of variational problems

$$(\partial_t V, \partial_t U)_{L^2(Q_i)} + (\mathbf{a}\boldsymbol{\nabla} V, \boldsymbol{\nabla}\partial_t U)_{L^2(Q_i)} - (F(\cdot, V, \boldsymbol{\nabla} V), \partial_t U)_{L^2(Q_i)} = (b, \partial_t U)_{L^2(Q_i)} \ . \tag{6.7}$$

for each *space-time slab* $Q_i = \Omega \times (t_i, t_{i+1})$ with Dirichlet values

$$V\big|_{Q_i}(\mathbf{x}, t_i) = V\big|_{Q_{i-1}}(\mathbf{x}, t_i) \ .$$

Aziz and Monk[6] provide a stability and convergence analysis of the method and show that several known time discretization schemes are recovered by applying different quadrature rules to (6.7).

6.2.2 Discontinuous Galerkin Methods

A different class of discretizations is based on a discontinuous Galerkin (dG) approximation[43] in time. The idea of combining a discontinuous-in-time approximation with a continuous spatial approximation seems to have been first introduced by Jamet[92] in 1978 with the goal to simplify the handling of variable domains. The starting point of the derivation is the weak formulation of equation (6.4) in space which we integrate over $(0,T)$ to obtain

$$\begin{aligned}\int_0^T (\partial_t V, U)_{L^2(\Omega)} \, \mathrm{d}t + \int_0^T (\mathbf{a}\boldsymbol{\nabla} V, \boldsymbol{\nabla} U)_{L^2(\Omega)} \, \mathrm{d}t - \\ \int_0^T (F(\cdot, V, \boldsymbol{\nabla} V), U)_{L^2(\Omega)} \, \mathrm{d}t = \int_0^T (b, U)_{L^2(\Omega)} \, \mathrm{d}t\end{aligned} \tag{6.8}$$

for $V, U \in \mathcal{C}^1((0,T), \mathbb{Y}^{\text{space}})$. In order to state the dG approximation of equation (6.8) we define $\mathbb{Y}^{\text{time}} = \prod_i \mathbb{Y}_i^{\text{time}}$ where $\mathbb{Y}_i^{\text{time}}$ is a local approximation space on (t_i, t_{i+1}). With this definition the discontinuous Galerkin weak formulation reads as: Find $V \in \mathbb{Y}$ so that

$$\begin{aligned}-(V, \partial_t U)_{L^2(Q_i)} + \left(\hat{V}, U\right)_{L^2(\Omega)} \Big|_{t_i}^{t_{i+1}} + \\ (\mathbf{a}\boldsymbol{\nabla} V, \boldsymbol{\nabla} U)_{L^2(Q_i)} - (F(\cdot, V, \boldsymbol{\nabla} V), U)_{L^2(Q_i)} = (b, U)_{L^2(Q_i)} \ ,\end{aligned} \tag{6.9}$$

for all test functions $U \in \mathbb{Y}$ on all space-time slabs Q_i. The choice of the *trace* $\hat{V}$ defines the type of the dG approximation. We use the definition

$$\hat{V}(t_i) = \begin{cases} V(0) & \text{if } t_i = 0 \, , \\ \lim_{t \nearrow t_i} V(t) & \text{otherwise.} \end{cases} \tag{6.10}$$

This method has been analyzed by Delfour et al.[54] for the discretization of ordinary differential equations and by Eriksson et al.[58], Jamet[92] for parabolic problems.

With the notation $V^+(t) = \lim_{s \nearrow t} V(s)$ we can rewrite equation (6.9) as

$$\begin{aligned}-(V, \partial_t U)_{L^2(Q_i)} + \left(V^+(t_{i+1}), U\right)_{L^2(\Omega)} + \\ (\mathbf{a}\boldsymbol{\nabla} V, \boldsymbol{\nabla} U)_{L^2(Q_i)} - (F(\cdot, V, \boldsymbol{\nabla} V), U)_{L^2(Q_i)} = (b, U)_{L^2(Q_i)} + \left(V^+(t_i), U\right)_{L^2(\Omega)} \ .\end{aligned} \tag{6.11}$$

An alternative derivation of equation (6.11) is obtained by applying integration by parts to equation (6.8) and adding the jump term

$$([V], U)_{L^2(\Omega)} = \left(\lim_{t \swarrow t_i} V(t) - V^+(t_i), U\right)_{L^2(\Omega)}$$

which penalizes weak discontinuities across slab boundaries.

Space-time discretizations using ansatz (and test) functions that are discontinuous in time have been used, for example, in the context of elastodynamics by Hughes and Hulbert[85], Hulbert and Hughes[86] and by Sathe et al.[140], Tezduyar et al.[154] for computational fluid dynamics on moving domains as well as for fluid-structure interaction.

6.2.3 Discretization on Non-Conforming Meshes

The tensor structure of the space-time approximation space $\mathbb{Y}$ has not been used in the derivation of equation (6.11). Therefore it is straightforward to extend the discretization to arbitrary approximation spaces built on a, potentially non-conforming, mesh on the space-time slab $Q_i = \Omega \times (t_i, t_{i+1})$. We can use the techniques discussed in Chapter 4 and Chapter 5 to construct $(d+1)$-dimensional meshes on the space-time slab Q_i that are used to the build trial space in equation (6.11).

In Algorithm 6.1 we sketch an algorithm for the solution of equation (6.4) using non-conforming space-time meshes. Since the proposed technique is applicable to a large variety of problems, many details have been left deliberately unspecified here.

```
t ← 0 and i ← 1
while t < T do
    Construct coarse initial mesh 𝒯_τ on Q_i
    loop
        Solve equation (6.11) for V ∈ 𝕐 for an appropriate subspace 𝕐 ⊊ 𝕏_τ
        built on the non-conforming space-time mesh 𝒯_τ
        Estimate the local error
        if total error small enough then
            break
        end if
        𝒯_τ ← refine(𝒯_τ)
    end loop
    t ← t + (t_{i+1} − t_i) and i ← i + 1
end while
```

Algorithm 6.1. *Time integration algorithm (schematic).*

6.2.4 Space-Time Transfer Operator

For solving equation (6.11) we need to assemble the boundary term $\left(V^+(t_i), U\right)_{L^2(\Omega)}$. Here, $V^+(t_i) = \lim_{\varepsilon \searrow 0} V(t_i - \varepsilon)$ is the restriction of the function V defined on the last space-time slab Q_{i-1} to $\Omega \times \{t_i\}$. Thus, $V^+(t_i) = \sum_\alpha V_\alpha \pi_\alpha^{i-1}$ where $\{\pi_\alpha^{i-1}\}$ denote the set of basis functions of the ansatz space on Q_{i-1} which do not vanish on the boundary. The test function U in contrast is defined on the space-time slab Q_i and thus expanded as $U = \sum_\beta U_\beta \pi_\beta^i$. Hence,

$$\left(V^+(t_i), U\right)_{L^2(\Omega)} = \sum_{\alpha,\beta} V_\alpha U_\beta \left(\pi_\alpha^{i-1}, \pi_\beta^i\right)_{L^2(\Omega)} . \tag{6.12}$$

The resulting mass matrix on the right-hand side cannot be assembled with the strategy discussed in Section 5.3.3 since two different finite element spaces are involved. In the following we use the

approximation

$$\left(V^+(t_i), U\right)_{L^2(\Omega)} \approx \sum_{\alpha,\beta} (\mathbf{T}V)_\alpha U_\beta \left(\pi^i_\alpha, \pi^i_\beta\right)_{L^2(\Omega)} , \tag{6.13}$$

where $\mathbf{T}$ maps between the boundary trace spaces on $\Omega \times \{t_i\}$ induced by the ansatz spaces on Q_{i-1} and Q_i, respectively. We use a local transfer operator (on the boundary) as discussed in Section 5.4.4.

When using uniform space-time meshes, (or more generally if the target and source boundary meshes are nested), the formulations (6.12) and (6.13) are equivalent. For general space-time adaptive meshes, however, the map $\mathbf{T}$ introduces an error that depends on how well $V^+(t_i)$ can be approximated on the boundary mesh induced by the mesh $\mathscr{T}_{\boldsymbol{\tau}}$ on Q_i. One possible approach to guarantee a sufficient approximation quality of the boundary mesh is the use of weighted error indicators in order to promote mesh refinement towards the lower boundary of the space-time slab. This approach is discussed further in Section 6.3.5.

6.2.5 Discretization of Monodomain and Bidomain equations

For the sake of completeness, in the following we state the space-time discretization of the bidomain (in parabolic-parabolic and parabolic-elliptic form) and monodomain equations.

We denote the stiffness matrices corresponding to the three bilinear forms $(\mathbf{G}_\mathrm{i}\boldsymbol{\nabla}\cdot, \boldsymbol{\nabla}\cdot)_{L^2(Q_i)}$, $(\mathbf{G}_\mathrm{e}\boldsymbol{\nabla}\cdot, \boldsymbol{\nabla}\cdot)_{L^2(Q_i)}$ and $(\mathbf{G}_\mathrm{mono}\boldsymbol{\nabla}\cdot, \boldsymbol{\nabla}\cdot)_{L^2(Q_i)}$ by $\mathbf{A}_\mathrm{i}, \mathbf{A}_\mathrm{e}$ and $\mathbf{A}_\mathrm{mono}$, respectively. Note that these differ from the $(d+1)$-dimensional Laplacian since $\boldsymbol{\nabla}$ does not contain the time derivative. In particular, these operators have a non-trivial kernel in $H^1(Q_i)/\mathbb{R}$.

Further, by $\mathbf{M}^+$ and $\mathbf{M}^-$ we denote the surface mass matrices on the upper and lower side of the space-time slab. We define $\boldsymbol{\Sigma}'$ as the stiffness matrix corresponding to the non-symmetric bilinear form $(\cdot, \partial_t \cdot)_{L^2(Q_i)}$ and $\boldsymbol{\Sigma} = -\boldsymbol{\Sigma}' + \mathbf{M}^+$. In the following $\mathbf{M}$ denotes the $(d+1)$-dimensional mass matrix.

With these definitions, the space-time discretization of the bidomain and monodomain equation reads as follows.

Bidomain equation (parabolic-parabolic). Solve the non-linear system $\mathbf{F}\left(\varphi_\mathrm{i}^{i+1}, \varphi_\mathrm{e}^{i+1}, \mathbf{s}^{i+1}\right) = \mathbf{b}^i$ with

$$\begin{aligned}
\mathbf{F}(\varphi_\mathrm{i}, \varphi_\mathrm{e}, \mathbf{s}) = &\left(\begin{bmatrix} C_\mathrm{m}\boldsymbol{\Sigma} & -C_\mathrm{m}\boldsymbol{\Sigma} & \mathbf{0} \\ -C_\mathrm{m}\boldsymbol{\Sigma} & C_\mathrm{m}\boldsymbol{\Sigma} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\Sigma} \end{bmatrix} + \frac{1}{\chi} \begin{bmatrix} \mathbf{A}_\mathrm{i} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_\mathrm{e} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \right) \begin{bmatrix} \varphi_\mathrm{i} \\ \varphi_\mathrm{e} \\ \mathbf{s} \end{bmatrix} , \\
&+ \begin{bmatrix} \mathbf{M} & -\mathbf{M} & \mathbf{0} \\ -\mathbf{M} & \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{bmatrix} I_\mathrm{ion}(\varphi_\mathrm{i} - \varphi_\mathrm{e}, \mathbf{s}) \\ \mathbf{0} \\ -\mathbf{Z}(\varphi_\mathrm{i} - \varphi_\mathrm{e}, \mathbf{s}) \end{bmatrix} , \\
\mathbf{b}^i = &\begin{bmatrix} \mathbf{M}^- & -\mathbf{M}^- & \mathbf{0} \\ -\mathbf{M}^- & \mathbf{M}^- & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M}^- \end{bmatrix} \begin{bmatrix} C_\mathrm{m}\mathbf{T}\varphi_\mathrm{i}^i \\ C_\mathrm{m}\mathbf{T}\varphi_\mathrm{e}^i \\ \mathbf{T}\mathbf{s}^i \end{bmatrix} + \begin{bmatrix} \mathbf{M} & -\mathbf{M} & \mathbf{0} \\ -\mathbf{M} & \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{bmatrix} I_\mathrm{app} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} .
\end{aligned} \tag{6.14}$$

Bidomain equation (parabolic-elliptic). Solve the non-linear system $\mathbf{F}\left(V^{i+1}, \varphi_{\mathrm{e}}^{i+1}, \mathbf{s}^{i+1}\right) = \mathbf{b}^{i}$ with

$$\begin{aligned}
\mathbf{F}\left(V, \varphi_{\mathrm{e}}, \mathbf{s}\right) &= \left(\begin{bmatrix} C_{\mathrm{m}}\boldsymbol{\Sigma} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\Sigma} \end{bmatrix} + \frac{1}{\chi} \begin{bmatrix} \mathbf{A}_{\mathrm{i}} & \mathbf{A}_{\mathrm{i}} & \mathbf{0} \\ -\mathbf{A}_{\mathrm{i}} & (\mathbf{A}_{\mathrm{i}} + \mathbf{A}_{\mathrm{e}}) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \right) \begin{bmatrix} V \\ \varphi_{\mathrm{e}} \\ \mathbf{s} \end{bmatrix} \\
&\qquad + \begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{bmatrix} I_{\mathrm{ion}}\left(V, \mathbf{s}\right) \\ \mathbf{0} \\ -\mathbf{Z}\left(V, \mathbf{s}\right) \end{bmatrix}, \\
\mathbf{b}^{i} &= \begin{bmatrix} \mathbf{M}^{-} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M}^{-} \end{bmatrix} \begin{bmatrix} C_{\mathrm{m}}\mathbf{T}V^{i} \\ \mathbf{0} \\ \mathbf{T}\mathbf{s}^{i} \end{bmatrix} + \begin{bmatrix} \mathbf{M} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{bmatrix} I_{\mathrm{app}} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}.
\end{aligned} \tag{6.15}$$

Monodomain equation. Solve the non-linear system $\mathbf{F}\left(V^{i+1}, \mathbf{s}^{i+1}\right) = \mathbf{b}^{i}$ with

$$\begin{aligned}
\mathbf{F}\left(V, \mathbf{s}\right) &= \left(\begin{bmatrix} C_{\mathrm{m}}\boldsymbol{\Sigma} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma} \end{bmatrix} + \frac{1}{\chi} \begin{bmatrix} \mathbf{A}_{\mathrm{mono}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \right) \begin{bmatrix} V \\ \mathbf{s} \end{bmatrix} + \begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{bmatrix} I_{\mathrm{ion}}\left(V, \mathbf{s}\right) \\ -\mathbf{Z}\left(V, \mathbf{s}\right) \end{bmatrix}, \\
\mathbf{b}^{i} &= \begin{bmatrix} \mathbf{M}^{-} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^{-} \end{bmatrix} \begin{bmatrix} C_{\mathrm{m}}\mathbf{T}V^{i} \\ \mathbf{T}\mathbf{s}^{i} \end{bmatrix} + \begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{M} \end{bmatrix} \begin{bmatrix} I_{\mathrm{app}} \\ \mathbf{0} \end{bmatrix}.
\end{aligned} \tag{6.16}$$

Note that these equations are conceptually very similar to the implicit Euler discretization of the bidomain and monodomain equation as discussed in Section 2.3.2.

6.3 Results

In the following we present the results of extensive numerical studies of the performance of the proposed space-time adaptive scheme. In Sections 6.3.1–6.3.4 we study the solution of the heat equation in $(1+1)$, $(2+1)$ and $(3+1)$ dimensions with the goal to (a) assess the additional gain that is possible through space-time adaptivity on top of a spatially adaptive discretization with the method of lines, (b) to study the behavior of linear solvers for the space-time discrete problem and (c) to show the feasibility of this approach even for large-scale $(3+1)$-dimensional problems. In Sections 6.3.5 and 6.3.6 we study the space-time discretization of the monodomain equation using the Bernus membrane model. In contrast to the linear heat equation, the space-time discretization of the monodomain equation leads to a coupled non-linear system of equations with six degrees of freedom per mesh node. We study the behavior of the non-linear Newton solver and discuss the role of stabilization for the solution of the linear problems arising in the Newton iterations.

The experiments presented in the following have been performed using the simulation code discussed in Section 5.4. All results with reported timings have been performed on the Cray XE6 "Monte Rosa" at the Swiss National Supercomputing Centre, featuring dual-socket nodes with AMD Interlagos CPUs, 32 GiB main memory per node and a Gemini interconnect. To avoid a

negative impact of the shared floating point units in the Bulldozer microarchitecture, in all experiments we placed only one process per Bulldozer module. The code was compiled with gcc-4.7.2 on this system.

6.3.1 (1+1)-dimensional Heat Equation

We consider the implicit Euler and space-time finite element discretization of the $(1+1)$-dimensional heat equation

$$\partial_t V - \Delta V = b \tag{6.17}$$

for $V \in \mathcal{C}^1\left((0,T),\mathcal{C}^2(\Omega)\right)$. For our experiments, we chose the right-hand side b such that the analytical solution of equation (6.17) is given by

$$V^*(x,t) = A \cdot \exp\left(-\frac{(x-\gamma(t))^2}{2\sigma^2}\right) \tag{6.18}$$

with $\gamma(t) = r \cdot \cos(2\pi \cdot t/p)$. Here, we used the parameters $A = 1$, $\sigma = 5 \cdot 10^{-2}$, $r = 0.5$ and $p = 5$. We solved the heat equation on the domain $\Omega = (-1,1)$ from time zero to $T = p = 5$. A contour plot of the exact solution is shown in Figure 6.2.

In the following we analyze the convergence of uniform and adaptive implicit Euler and space-time finite elements approximations of (6.18) in the $L^2(H^1)$-semi-norm which is defined by

$$|U|^2_{L^2(H^1)} = (\nabla U, \nabla U)_{L^2(Q)} = \int_0^T (\nabla U, \nabla U)_{L^2(\Omega)} \, \mathrm{d}t \; . \tag{6.19}$$

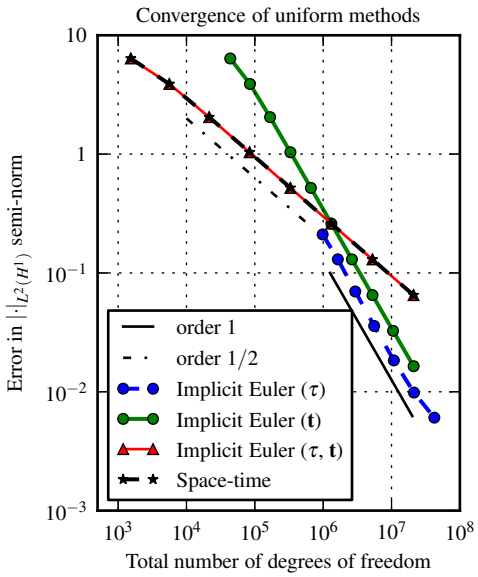
We chose the $L^2(H^1)$-norm for the following experiments because on the one hand it is an appropriate semi-norm for steering the adaptive refinement process and on the other hand it can be computed accurately for both the implicit Euler time discretization and the space-time discretization. In the former case we use a summed trapezoidal rule to incrementally compute the time integral in equation (6.19).

In the following experiments we used continuous ansatz and test functions in $\mathbb{Y}^c_{\boldsymbol{\tau}}$ built on meshes $\mathscr{T}_{\boldsymbol{\tau}}$. Note we used point-wise constraints but did not restrict the coarse-to-fine ratio in our tests. We started from an initial tessellation of Ω consisting of two equally spaced elements and set $\delta_\ell = 1/4$ for all levels ℓ. Thus, spatial refinement was achieved solely by refining the trees $\boldsymbol{\tau}$. The space-time slabs had length $\frac{1}{2}$ in time direction.

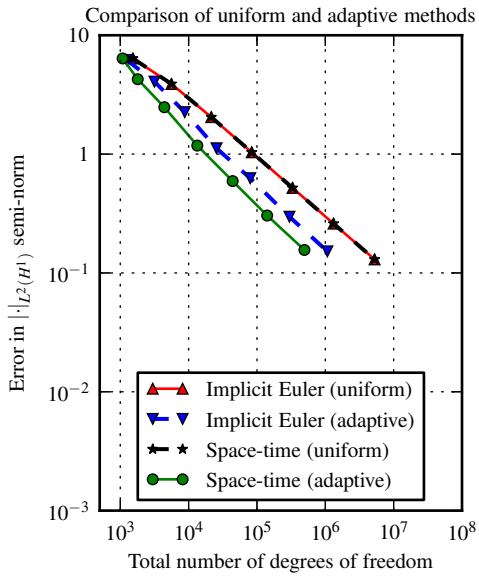
In Figure 6.1 the results of the convergence studies are shown. Figure 6.1a shows the measured error $|V_{\boldsymbol{\tau}} - V^*|_{L^2(H^1)}$ plotted against the total number of degrees of freedom when using uniform refinement in time, in space or both. The total number of degrees of freedom equals

$$\#\,\text{dofs} = \begin{cases} \sum_{\text{laps}} (L_{\text{lap}} + 1) \dim \mathbb{Y}^c_{\boldsymbol{\tau}} & \text{(implicit Euler)} \\ \sum_{\text{slabs}} \dim \mathbb{Y}^c_{\boldsymbol{\tau}} & \text{(space-time discretization)} \end{cases} .$$

When refining in time (with a fixed but very fine spatial discretization) or in space (with a fixed by small time step size τ) we can observe first order convergence as one expects from the first-order implicit Euler method and from finite element convergence theory. For uniform refinement



(a) *Convergence of uniform discretizations.*



(b) *Comparison of convergence of uniform and adaptive discretizations.*

Figure 6.1. *Convergence of uniform and adaptive discretizations for the approximation of the $(1+1)$-dimensional heat equation. In the left plot we vary both the time step size τ and the spatial resolution (controlled by the depth of the trees $\boldsymbol{\tau}$).*

Table 6.1. *Comparison of the total number of degrees of freedom and the measured error in the $|\cdot|_{L^2(H^1)}$ semi-norm for a uniform implicit Euler discretization, a spatially adaptive and a space-time adaptive discretization.*

# dof	Error
Implicit Euler (uniform)	
1,530	6.3793
5,610	3.8925
21,450	2.0470
83,850	1.0359
331,530	0.5195
1,318,410	0.2600
5,258,250	0.1300

# dof	Error
Implicit Euler (adaptive)	
1,242	6.1987
3,162	4.0703
8,778	2.2530
25,610	1.1195
79,722	0.6267
298,634	0.2972
1,072,170	0.1519

# dof	Error
Space-Time (adaptive)	
1,082	6.3798
1,802	4.2657
4,410	2.4807
13,402	1.1796
44,074	0.5927
141,034	0.3040
494,970	0.1560

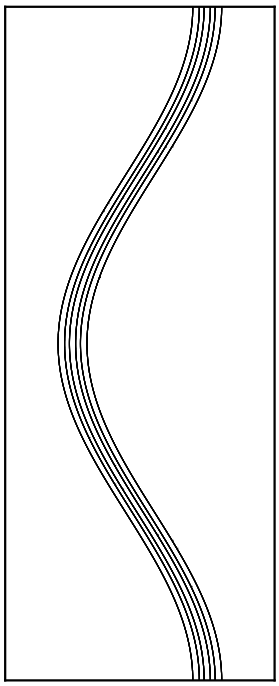


Figure 6.2. *Contours of the exact solution V^*. The vertical axis equals the time.*

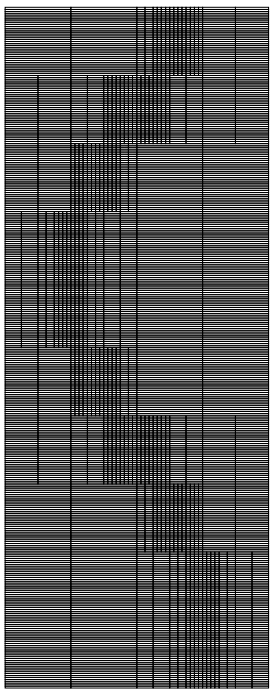


Figure 6.3. *Space-time representation of the spatially adapted mesh. For the visualization the mesh is downsampled by a factor of two in each direction.*

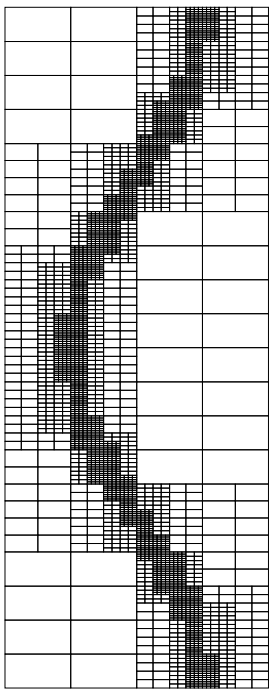


Figure 6.4. *Space-time adaptive mesh. For the visualization the mesh is downsampled by a factor of two in each direction.*

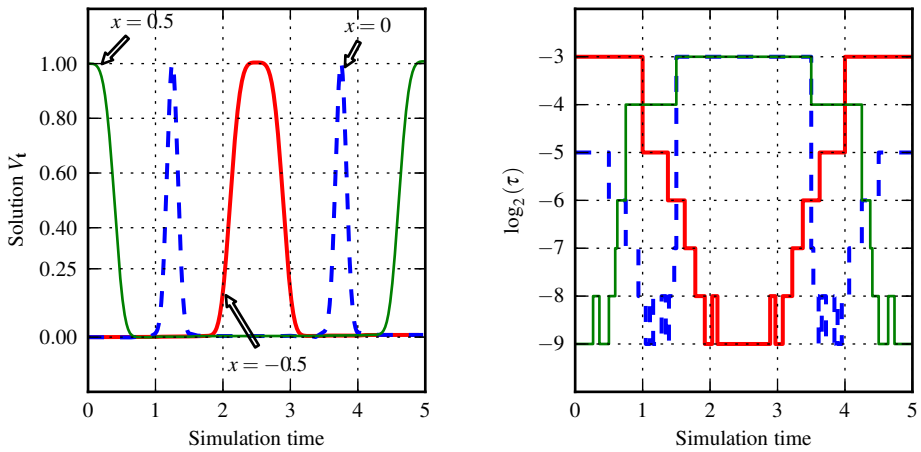


Figure 6.5. *Plot of the numerical solution V_τ (left) and the corresponding local time steps τ for $x = -0.5$, $x = 0$ and $x = 0.5$.*

in space and time we measure the expected convergence rate of $1/2$ for both the method of lines discretization with an implicit Euler method and the space-time discretization. Note that a scaling of $(\#\,\text{dofs})^{1/2}$ is equivalent to first-order convergence when substituting the mesh width for the number of degrees of freedom. From Figure 6.1a we can see that the method of lines approach and the space-time discretization lead to almost identical errors.

In Figure 6.1b we compare the error measured with uniform discretizations (same data as used for Figure 6.1a) to the error measured with a spatially adaptive method (using a fixed time step size) and an adaptive space-time discretization. The adaptive refinement process started from a coarse mesh in each new lap (on each new time slab, respectively). In each repetition, all leaves eligible for refinement with a local $L^2(H^1)$-error within 25% of the maximum error among all eligible leaves were marked for refinement. A leaf is eligible for refinement if its level is smaller than the maximally allowed tree depth. The mesh adaptation was stopped once the error was within 25% of the error measured with a uniformly refined discretization corresponding to the maximal allowed tree depth. The results presented in Figure 6.1b show that spatially adaptivity can provide a significant reduction in the number of degrees of freedom required to achieve a given error tolerance. Using a space-time discretization to enable local time stepping can provide an additional reduction in degrees of freedom with respect to a spatially adaptive discretization. In Table 6.1 the numbers of degrees of freedom are presented alongside the measured discretization error. Using a spatially adaptive discretization, the cost (defined as the quotient of error and the number of degrees of freedom) can be reduced by a factor of up to 5.6. By employing a space-time discretization this factor can be improved to 12.7.

Let us point out that in the experiments with the spatially adaptive discretization we increased L_{lap} in proportion to the decrease in τ so that a lap corresponds to a fixed interval of length $\frac{1}{2}$ in $(0,T)$. Another approach would be to keep L_{lap} fixed which leads to lower dimensional approximation spaces but also requires more frequent re-assembly of the system matrices. The additional reduction in the number of degrees of freedom is however low compared to the reduction achieved through local time stepping.

In Figure 6.5 the values of the solution $V_{\boldsymbol{\tau}}(x,t)$ and the local time step $\tau(x,t)$ along the three lines $\{x=-0.5\}$, $\{x=0\}$ and $\{x=0.5\}$ are shown. In this example, the time step size varies by a factor of $2^6=64$ between the coarsest and the finest step size and (in contrast to global time step control) the employed local time step is adjusted to the *local* behavior of the solution.

Figure 6.3 and Figure 6.4 show the space-time mesh used for the method of lines discretization and for the space-time discretization. While both approaches allow for tracking the Gaussian peak with finer meshes, only the space-time mesh can coarsen the mesh in time at points x where the solution is nearly constant.

6.3.2 Stabilization of the Space-Time Mortar Element Method

Jamet[92] proved first order convergence of the finite element discretization of the space-time heat equation on conforming meshes under the assumption $V \in H^2(Q) \cap C^0\left((0,T),H^2(\Omega)\right)$. In Section 6.3.1 we have experimentally verified the first order convergence for conforming ansatz spaces.

When dealing with a weak diffusion (i.e., $\|\mathbf{a}\| \ll 1$ in equation (6.4)), the space-time formulation becomes convection dominated due to the convective term $(V, \partial_t U)_{L^2(Q)}$. A standard approach for stabilization is, for example, the *streamline upwind Petrov-Galerkin* (SUPG) approach[30,34] which is based on the stabilization term

$$\sum_{E \in \mathscr{T}_\tau} \varepsilon_E \Big(\left(\partial_t V - \boldsymbol{\nabla} \cdot (\mathbf{a} \boldsymbol{\nabla} V) - F(\mathbf{x}, V, \boldsymbol{\nabla} V) - b\right), \partial_t U \Big)_{L^2(E)}$$

that is added to the left hand side of equation (6.9). Here, $\varepsilon > 0$ is an element-wise constant stabilization parameter.

The results from Section 6.3.1 show that stabilization is not necessary for the solution of the isotropic heat equation (with conductivity coefficient 1) when using continuous ansatz functions. Unexpectedly, though, the discretization using the non-conforming mortar approximation space $\mathbb{Y}^{\mathrm{m}}_\tau$ shows strong oscillations even when employed on an uniform mesh, see Figure 6.6a. As mentioned in Section 5.5.2 the mortar discretization on an uniform mesh is not equivalent to a conforming discretization since we do not enforce continuity on the wire basket.

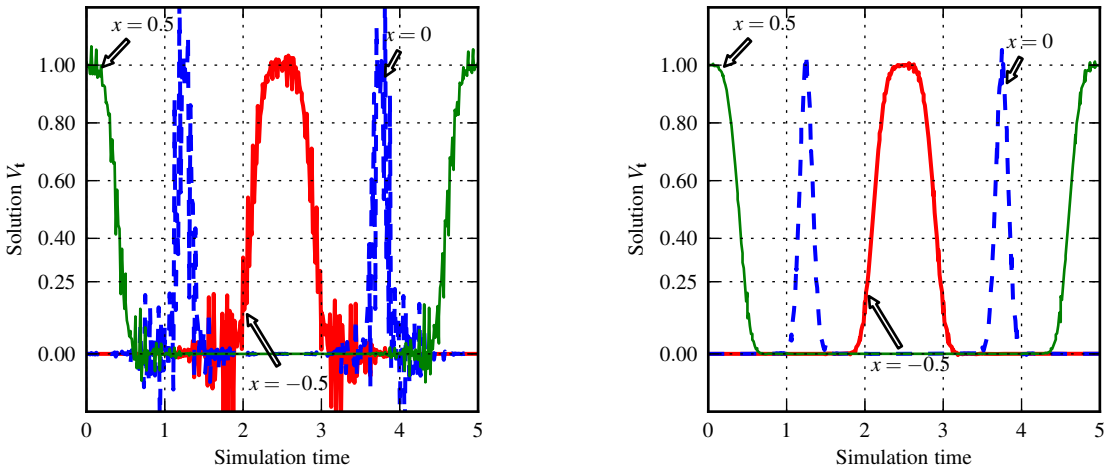
Oscillations can be observed for both large and small diffusion coefficients which indicates that they are not caused by a dominant first order term. We have verified these results with a second implementation to rule out programming errors as the source of this instability. In Figure 6.6b the discrete solution of the stabilized space-time heat equation using a mortar discretization is shown. This result was obtained by adding an additional diffusion term

$$\sum_{E \in \mathscr{T}_\tau} \frac{1}{2} \operatorname{diam}(E) \, (\partial_t V, \partial_t U)_{L^2(E)} \tag{6.20}$$

to the left hand side of the discretized equation. Initial experiments with a SUPG discretization showed that the modification of the right-hand side lead to large overshoots at the upper boundary $\{t_{i+1}\} \times \Omega$ of the space-time slab.

The stabilization term (6.20) introduces a weak artificial diffusion along the time axis. Since the bilinear form $(\mathbf{a}\boldsymbol{\nabla}\cdot, \boldsymbol{\nabla}\cdot)_{L^2(Q)}$ vanishes for all functions with zero spatial gradient (but arbitrary time evolution) it is not coercive and the standard mortar element theory cannot be applied. By adding (6.20) we replace the spatial diffusion by a strongly anisotropic space-time diffusion which aids the stability of the discretization.

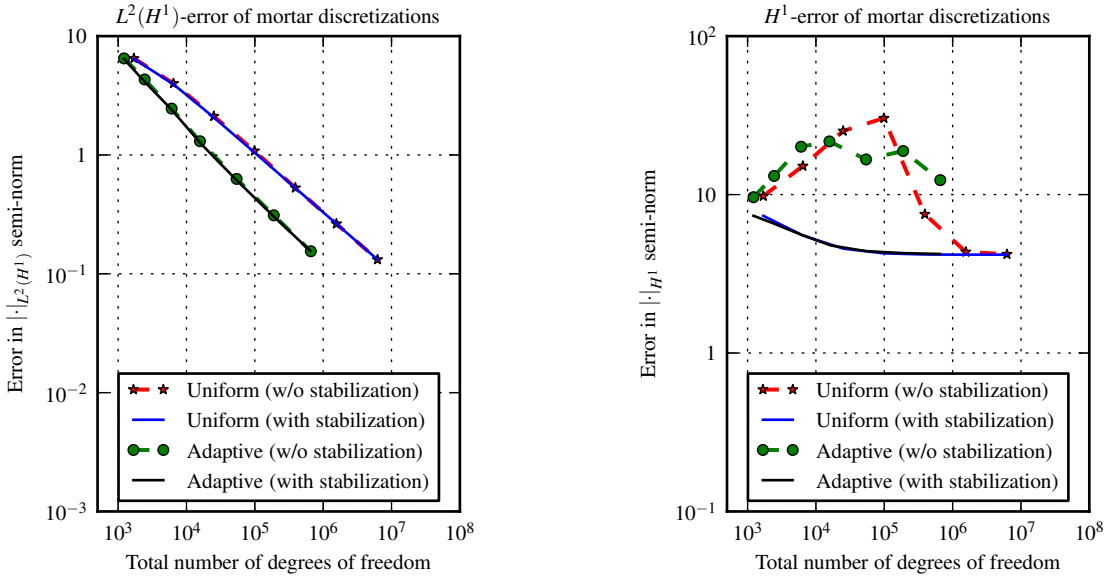
Since the observed oscillations only occur along the time axis, the error of the discrete solution $V_\tau \in \mathbb{Y}^{\mathrm{m}}_\tau$ in the $|\cdot|_{L^2(H^1)}$ semi-norm differs only slightly between stabilized and non-stabilized discretization, see Figure 6.7a. The measured error is in both cases of the same order as the error of a conforming discretization, cf. Figure 6.1b. Since the adaptive refinement is driven by the local $L^2(H^1)$-error per tree leave, we can observe a similar reduction in the degrees of freedom as reported in Section 6.3.1. When considering the error in the $|\cdot|_{H^1(Q)}$ semi-norm a strong reduction in the measured error is observed due to the stabilization, see Figure 6.7b. Note that the space-time discretization of the heat equation with conforming or non-conforming ansatz spaces is not convergent in the H^1-norm.



(a) *Mortar element solution without stabilization.*

(b) *Mortar element solution with stabilization.*

Figure 6.6. *Comparison of the discrete solution $V_\tau \in \mathbb{Y}^{\mathrm{m}}_\tau$ without (left plot) and with (right plot) stabilization.*



(a) *Error of the mortar element solution in the $|\cdot|_{L^2(H^1)}$ semi-norm.*

(b) *Error of the mortar element solution in the $|\cdot|_{H^1}$ semi-norm.*

Figure 6.7. *Comparison of the error of the mortar element solution in the $L^2(H^1)$- and $H^1(Q)$-semi-norms with and without stabilization.*

6.3.3 (2+1)-dimensional Heat Equation

In this section we study the solution of the $(2+1)$-dimensional heat equation via space-time finite elements. As in Section 6.3.1, we chose the right-hand side such that the analytical solution is given by a moving Gaussian

$$V^*(\mathbf{x},t) = A \cdot \exp\left(-\frac{|\mathbf{x}-\boldsymbol{\gamma}(t)|^2}{2\sigma^2}\right) \tag{6.21}$$

with

$$\boldsymbol{\gamma}(t) = r\Big(\cos(2\pi \cdot t/p)\,\mathbf{e}_1 + \sin(2\pi \cdot t/p)\,\mathbf{e}_2\Big)\,.$$

As in Section 6.3.1, we set $A = 1$, $\sigma = 5 \cdot 10^{-2}$, $r = 0.5$ and $p = 5$. We solved the heat equation on the ball $B_1(0)$ from time zero to $T = p = 5$.

We consider the error in the $L^2(H^1)$-semi-norm and used the exact error to steer the adaptive refinement as in Section 6.3.1. In Figure 6.8 we show a comparison of the measured error for a uniform and adaptive implicit Euler and space-time discretization. As in the case of the $(1+1)$-dimensional heat equation we observe a large gain through the use of spatial adaptivity and an additional consistent improvement of $2\times$ or more through the use of space-time finite elements. From the quotient of the number of degrees of freedom and the measured error, as reported in Table 6.2 for different resolutions, we see that a gain of up to 26 is possible through spatial adaptivity and a gain of up to 66 using space-time finite elements.

In Figure 6.10 and Figure 6.11 the contours of the numerical solution of the heat equation using space-time finite elements and an implicit Euler discretization are depicted. For the spatially adaptive simulation we only show the solution at the end of a lap (with the exception of the first slice which shows the initial conditions). Note that the same spatial mesh is used for all time steps in the same lap. Thus the resulting tensor mesh on $\Omega \times (t_i, t_{i+1})$ is not as sparse as the corresponding space-time mesh because the latter is not restricted to a tensor structure.

The space-time discretized heat equation differs structurally significantly from the corresponding spatial problem that results from a method of lines discretization. The arising linear system is non-symmetric and the discretization of the Laplacian is singular on $H^1(Q)/\mathbb{R}$ with all spatially constant functions in the kernel. Thus, efficient linear solvers and preconditioners for the spatial problem fail for the space-time problem. For example, we performed early experiments with an algebraic multi-grid[53,62], a highly efficient preconditioner for the elliptic spatial problem, without success. Block preconditioning can be used to precondition the space-time linear system. In the left plot in Figure 6.9 the number of iterations for the solution of the heat equation with approximately 10^6 total degrees of freedom is shown. Here, we used a GMRES(30) linear solver with a relative and absolute tolerance of 10^{-8} and a restricted additive Schwarz method from the PETSc[11,141] package with an overlap of two and UMFPACK[49–52] for the solution of the local problem. We used 64 processing elements for the implicit Euler discretization and one block per processing element. For the space-time finite element discretization we used 256 processing elements.

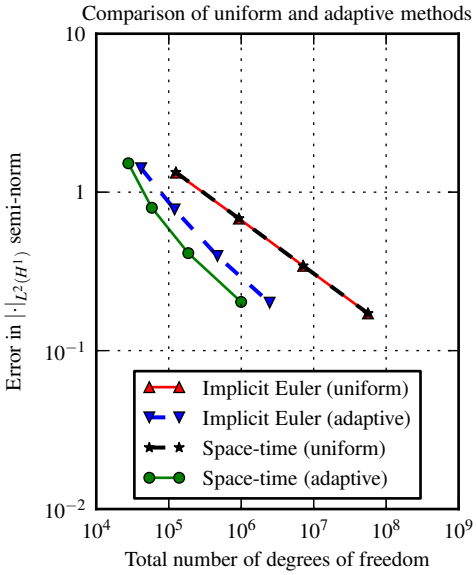


Figure 6.8. *Comparison of the convergence of uniform and adaptive discretizations of the* $(2+1)$*-dimensional heat equation.*

Table 6.2. *Quotient of the number of degrees of freedom (in millions) and the measured discretization error for the uniform and adaptive implicit Euler discretization and the adaptive space-time discretization of the* $(2+1)$*-dimensional heat equation. Each row corresponds to a data point from Figure 6.8.*

Implicit Euler (uniform)	Implicit Euler (adaptive)	Space-Time (adaptive)
0.09	0.03	0.02
1.37	0.15	0.07
20.89	1.19	0.45
327.08	12.31	4.92

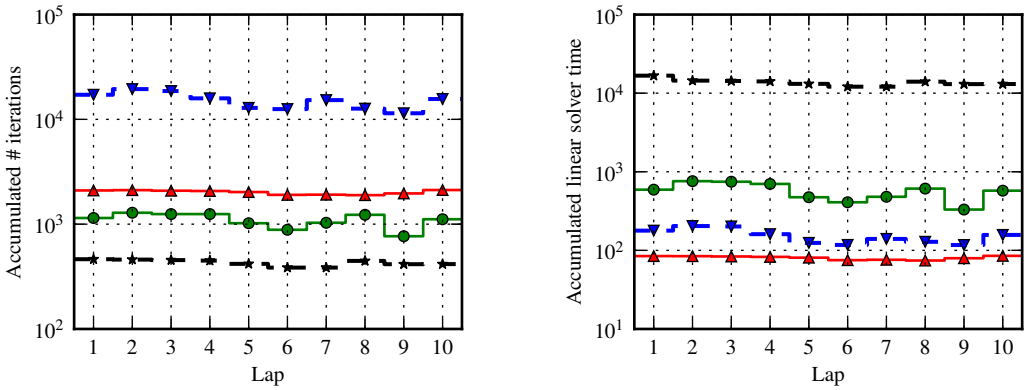


Figure 6.9. *Comparison of the linear solver performance. The left plot shows the accumulated number of iterations required by the GMRES solver with restricted additive Schwarz preconditioner. The right plot shows the total time spent in the solver. The line styles are the same as in Figure 6.8.*

Table 6.3. *Scaling behavior of a GMRES linear solver with restricted additive Schwarz preconditioner for an implicit Euler (top) and space-time (bottom) discretization.*

# levels	# iterations	solver time [s]
Implicit Euler (uniform)		
1	176	1.50 s
2	409	3.88 s
3	928	14.76 s
4	2,018	80.57 s

# levels	# iterations	solver time [s]
Implicit Euler (adaptive)		
1	390	3.15 s
2	1,192	9.86 s
3	4,253	38.62 s
4	15,148	152.84 s

# levels	# iterations	solver time [s]
Space-time (uniform)		
1	80	9.46 s
2	148	63.68 s
3	288	778.55 s
4	429	13,693.89 s

# levels	# iterations	solver time [s]
Space-time (adaptive)		
1	77	11.06 s
2	207	41.92 s
3	517	146.64 s
4	1,097	567.70 s

Table 6.4. *Scaling of a conjugate gradient solver with BoomerAMG preconditioner for an implicit Euler discretization.*

# levels	# iterations	solver time [s]
Implicit Euler (uniform)		
1	46	1.89 s
2	86	4.81 s
3	163	13.96 s
4	320	47.22 s

# levels	# iterations	solver time [s]
Implicit Euler (adaptive)		
1	170	4.68 s
2	496	16.52 s
3	1,460	53.72 s
4	4,025	168.60 s

For a uniform method of lines discretization with an implicit Euler time discretization scheme, the linear solver requires on average 31.5 iterations per time step and 2,018 iterations per lap (consisting of 640 time steps). For the space-time discretized problem, the linear solver converges on average within 429 iterations. As one would expect from a block preconditioner which is not optimal, the average number of linear solves for adaptive discretizations is lower than the number measured on a uniform mesh of the same minimal mesh width. For the adaptive discretizations we measure an average of 18.3 and 75.15 for the number of iterations per time step (for the implicit Euler discretization) or iterations per solution (for the space-time discretization), respectively. The average accumulated number of iterations per lap was found to be 15,148 and 1,097.1, respectively. Note that the systems solved per time step have varying dimensions due to the adaptive refinement procedure.

Despite the lower number of iterations measured for the space-time simulations the accumulated time spent in the linear solver is larger, see the right plot in Figure 6.9. We measure an accumulated linear solver time of 80.6 s for the implicit Euler method and 13,693.9 s for the space-time discretized method. The linear solver for the adaptive method of lines discretization is about $2\times$ slower than the solver on a uniform mesh (152.8 s per lap on average). The linear solver for the adaptive space-time discretization on the other hand is $24.12\times$ faster than the solver for the uniform space-time discretization (567.7 s per lap). In Table 6.3 the average number of iterations and the average time per lap is shown for different refinement levels. Each row in the table corresponds to one point in Figure 6.8. It is apparent that the solver for the space-time system shows a significantly worse scaling behavior in the solution time compared to the solver for the implicit Euler method. Since the increase in the solution time is disproportional to the increase in the number of solver iterations, it is most likely caused by the sub-optimal scaling of the local solver. For the implicit Euler method, the local problem sizes increase by $2^2 = 4$ whenever a new level is added. In contrast, the local problem sizes increase by $2^3 = 8$ for the space-time discretization. Moreover, the sparsity pattern of the stiffness matrix on the local block differs in the two cases. Since the performance of the sparse solver depends on the sparsity pattern of the stiffness matrix, the solution of a sequence of two-dimensional problems is not equivalent to the solution of a three-dimensional problem of the same size. In general, the latter will perform worse due to additional fill-in.

For comparison we also list the iteration counts and solver times for a method of lines discretization using a conjugate gradient solver with (untuned) algebraic multi-grid solver (using BOOMERAMG[53,62]) in Table 6.4. The algebraic multi-grid preconditioner gives level-independent convergence rates and hence the reported accumulated number of iterations increases approximately linearly with the level. Up to three levels of refinement, the solution time of the block preconditioned GMRES solver is comparable to that of the multi-grid preconditioned conjugate gradient. On adaptive meshes the block preconditioned GMRES solver is faster in all our measurements.

Let us point out that our presentation of the measured data has a slight bias towards the uniform mesh methods since we scale timings to a single processing elements assuming linear scaling. This is necessary in order to be able to compare timings obtained with a different number of processing elements. However, the assumption of linear scaling may be violated for the solution of the linear systems on the coarser meshes during the adaptive refinement procedure.

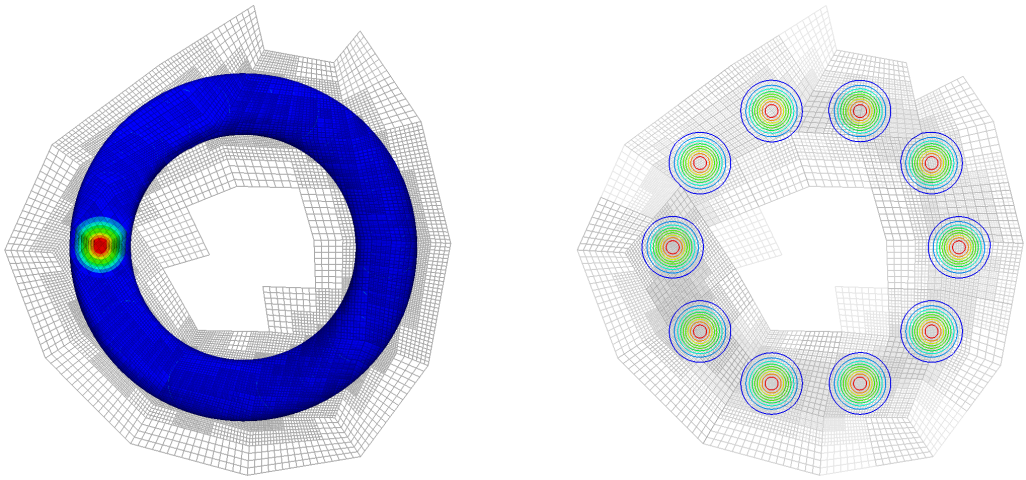


Figure 6.10. *Projected view of the contours of the discrete solution using space-time finite elements (left) and an implicit Euler time discretization (right).*

6.3.4 (3+1)-dimensional Heat Equation

To assess the feasibility of the solution of three-dimensional, time-dependent partial differential equations using space-time finite elements, in this section we consider the $(3+1)$-dimensional heat equation. We used the same analytic solution as in Section 6.3.3, i.e., a Gaussian peak moving in the xy plane. We used a structured mesh on the domain $\Omega = (-1,1)^3$, space-time slabs of extent $\frac{1}{2}$ in time direction, and a coarse tessellation of $2 \times 2 \times 2$ and $2 \times 2 \times 2 \times 1$ for the adaptive method of lines and adaptive space-time discretizations, respectively.

The results of our experiments are summarized in Figure 6.12 and Table 6.5. The measurements are in line with the experimental results from the previous sections, i.e., we observe a large reduction in the number of degrees of freedom by means of spatial adaptivity and an additional improvement by a factor of about two when adding local time stepping via space-time finite elements.

6.3.5 (1+1)-dimensional Monodomain Equation

In this section we study the space-time solution of the $(1+1)$-dimensional monodomain equation. In contrast to the heat equation studied in the previous section, the space-time discretization of the monodomain equation leads to a non-linear system of equations (see Section 6.2.5).

We consider the solution of the monodomain equation on the domain $\Omega = (-1,1)$ with a conductivity tensor $\mathbf{G}_{\text{mono}} = 2\ \text{mS/cm}$ and an applied current $I_{\text{app}} = 250\ \mu\text{A/cm}^2$ for $\frac{1}{4}$ ms in $\left(-\frac{1}{2}, \frac{1}{2}\right)$. For our experiments we used the Bernus membrane model.

In a first set of experiments we used conforming ansatz spaces on uniform spatial and space-time meshes with spatial mesh width $1/128$ cm, a step size of $1/64$ ms and a lap size $L_{\text{lap}} = 64$. The

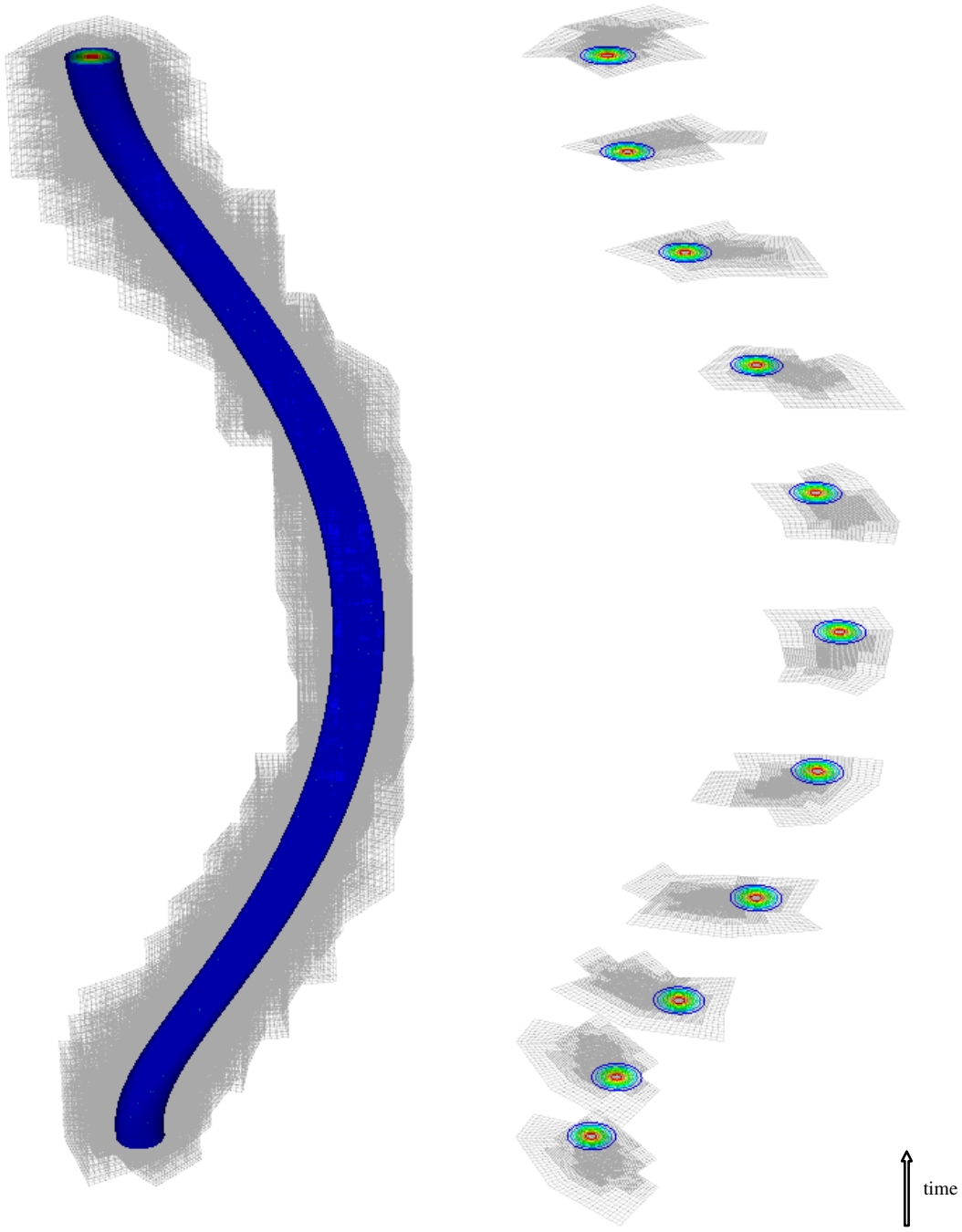


Figure 6.11. *Space-time contour plot of the discrete solution using space-time finite elements (left) and an implicit Euler time discretization (right). The wireframe of the mesh on leaves with level* ≥ 3 *is overlayed to indicate the structure of the adaptively refined meshes.*

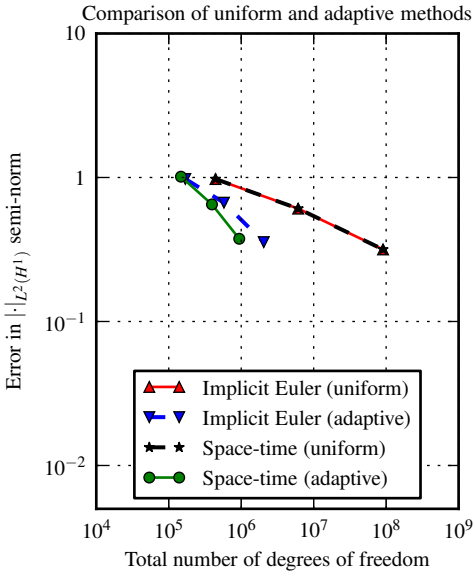


Figure 6.12. *Comparison of the convergence of uniform and adaptive discretizations of the* $(3+1)$*-dimensional heat equation.*

Table 6.5. *Quotient of the number of degrees of freedom (in millions) and the measured discretization error for the uniform and adaptive implicit Euler discretization and the adaptive space-time discretization of the* $(3+1)$*-dimensional heat equation. Each row corresponds to a data point from Figure 6.12.*

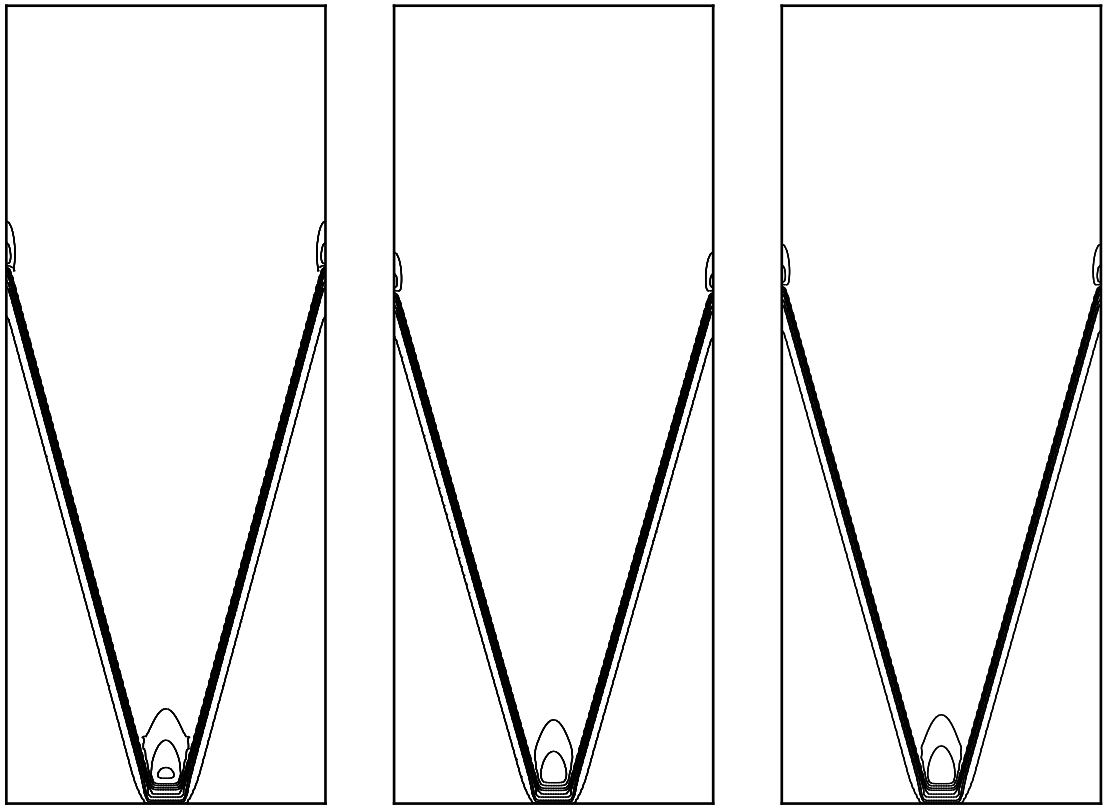
Implicit Euler (uniform)	Implicit Euler (adaptive)	Space-Time (adaptive)
0.45	0.17	0.15
10.05	0.86	0.61
286.35	5.75	2.50

mesh for the space-time discretization thus had an extent of 1 ms in the time direction. The linear system was solved with the direct solver MUMPS[3,4]. The non-linear system for the implicit Euler and space-time finite element discretization was solved by a Newton method with backtracking. The backtracking algorithm sequentially tested step sizes $1, 2^{-1}, 2^{-2}, 2^{-3}$ until the functional value is ≤ 1.05 times the previous value. If this criterion was not met, a step size of 2^{-3} was used. The Newton solver stopped if the residual norm was less than 10^{-8}.

A contour plot of the solution in space-time view on $(-1,1) \times (0,20)$ is shown in Figure 6.13. The implicit Euler and the space-time discretization give comparable results. The solution computed by the space-time discretization features a slightly slower depolarization front compared to the implicit Euler method and both, the implicit Euler and space-time solution, feature a slower depolarization time compared to the implicit-explicit Euler solution.

In Table 6.6 the number of Newton iterations and function evaluations for a selection of the time laps are shown. Due to the small time step size, the Newton solver for the method of lines discretization converges quickly in 2–3 iterations. By comparing the average number of iterations and the average number of function evaluations we can see that the backtracking is rarely invoked by the Newton solver, i.e., the step size 1 is used in most cases.

The number of Newton iterations required for the solution of the non-linear space-time system is between two and six times larger. The backtracking algorithm is invoked many times to calculate reduced step sizes, which indicates a “rougher” non-linearity. During the solution of the non-linear system on the second space-time slab, the backtracking algorithm fails four times, i.e., is unable



(a) *Implicit-explicit Euler method.* **(b)** *Implicit Euler method.* **(c)** *Space-time discretization.*

Figure 6.13. *Contours of the solution of the* $(1+1)$*-dimensional monodomain equation using an implicit-explicit Euler (left), implicit Euler (middle) and space-time (right) discretization. The vertical axis equals the time. For the visualization, the simulation domain has been scaled in time direction.*

to obtain a sufficient bound on the function value for the permissible step sizes. Nevertheless, the Newton solver converges in 14 iterations.

Since the one-dimensional stiffness matrix can be trivially inverted due to the tridiagonal structure a comparison of the solution times is not meaningful.

The time step size (explicitly through the choice of τ or through the mesh width in time direction) influences the strength of the non-linearity in the functional $\mathbf{F}$ that is solved in an implicit Euler step or on a space-time slab. Since the non-linearity is scaled by τ or by the space-time mass-matrix (the entries of which scale linearly in the mesh width in time-direction), the non-linear system is easier to solve on finer space-time meshes. In the case of a space-time discretization, one moreover expects the extent of the mesh in time-direction to influence the convergence behavior of the non-linear solver.

Table 6.6. *Number of Newton iterations and evaluations of the functional for a selection of the time laps. For the implicit Euler, average and accumulated numbers are shown.*

Lap	avg. # iters	acc. # iters	avg. # fct evals	acc. # fct evals
Implicit Euler				
1	2.9	187	6.8	438
2	2.9	187	6.8	438
5	2.9	187	6.8	438
10	2.9	188	6.9	440
15	2.0	128	5.0	320
17	1.4	90	3.8	244

Lap	# iterations	# function evaluations
Space-time		
1	6	15
2	14	55
5	7	17
10	8	21
15	8	21
17	8	21

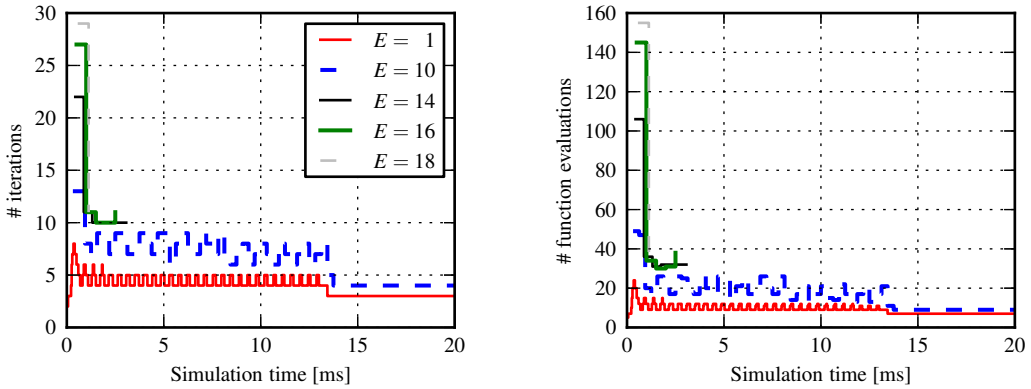


Figure 6.14. *Number of Newton iterations (left) and functional evaluations (right) in the dependence of the extent $E/64$ ms in time direction.*

In order to quantify this dependence we considered the discretization of the $(1+1)$-dimensional monodomain equation on a uniform space-time mesh with mesh width $1/64$ (cm and ms, respectively) and incrementally increased the number of elements $E \in \mathbb{Z}_{\geq 1}$ in time direction. Thus, the space-time slabs had extent $E/64$ ms in time direction. For these experiments we allowed step sizes $\{2^{-j}\}_{j=0}^{7}$ for the backtracking algorithm.

The number of Newton iterations and functional evaluations over time are plotted in Figure 6.14. Initially, the increase in number of iterations and functional evaluations is less than $2\times$. Starting at $E \geq 14$ however, the solver fails due to a floating point exception in the backtracking algorithm. When increasing the extent of the space-time mesh further, the failure occurs earlier. Starting with $E = 22$, the Newton solver is unable to solve the non-linear system on the first space-time slab.

Finally, we consider the space-time adaptive solution of the monodomain equation. In these

experiments we used a space-time slab of length 0.25 ms in time direction. Four refinement levels were allowed and we set $\delta_\ell = 1/4$ for all levels ℓ. The Newton solver terminated if the non-linear residual norm was below 10^{-8}. We used a GMRES(30) iterative solver with a block Jacobi preconditioner with four subdomains. The linear solver terminated after $1,000$ steps or if the absolute norm was below 10^{-10} or if the initial residual was reduced by 10^{-8}.

We employed a gradient-based error indicator

$$\left(\eta_o^\Sigma\right)^2 = \sum_{E \in \mathscr{T}_o} (\boldsymbol{\nabla} V_{\boldsymbol{\tau}}, \boldsymbol{\nabla} V_{\boldsymbol{\tau}})^2_{L^2(E)} \ .$$

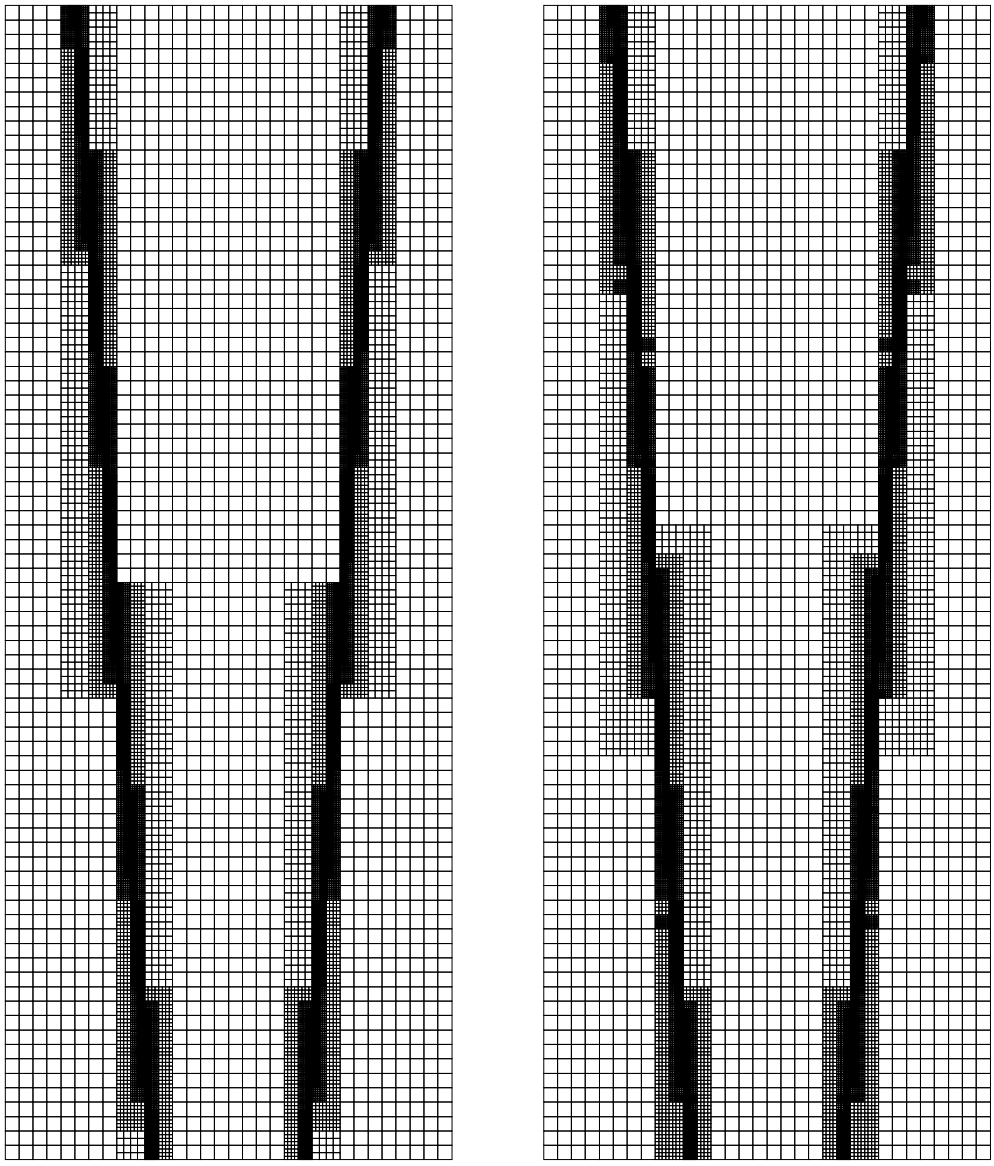
In previous experiments with the $(d+1)$-dimensional heat equation we used the exact error with respect to a known analytic solution. However, when using error indicators (in contrast to error estimators) to steer the adaptive refinement, the refinement process can potentially be misguided by "faulty" initial conditions. As discussed in Section 6.2.4 we use a space-time transfer operator between the trace spaces on the lower and upper boundary of Q_i to simplify the assembly of the jump term. The trace operator introduces an additional error if the target mesh is too coarse. In order to ensure a sufficient approximation property of the boundary mesh, we can promote refinement towards the lower boundary by assigning weights $w_o = 2$ to leaves touching the lower boundary. All other leaves are weighted with $w_o = 1$.

In each refinement step, the leaves o with $\left(w_o \eta_o^\Sigma\right) \geq \frac{1}{2} \max_{o'} \eta_{o'}^\Sigma$ were marked for refinement. Our refinement strategy stopped if no leaves were marked for refinement or if $\sum_o \eta_o^\Sigma \leq 10$. Note that we only used the weights when comparing against the maximal error indicator and thus the spectrum of the indicators was left unchanged. Therefore, we can guarantee that this modification does not reduce the approximation quality of the ansatz spaces in the interior of the space-time slab.

In Figure 6.15 a wireframe representation of the adaptive meshes on 20 space-time slabs on $(-1,1) \times (3.75, 7.75) \subset Q$ is shown. Figures 6.15a and 6.15b show the adaptive mesh with and without weighting, respectively.

Figure 6.16 shows the number of degrees of freedom of the adaptive ansatz spaces. We compare with a spatially adaptive implicit Euler discretization with step size $\tau = 1/256$ ms. During the depolarization phase, local time stepping reduces the accumulated number of degrees of freedom by a factor of approximately two. The use of the modified marking strategy increases the number of degrees of freedom by at most a factor of 1.24 but on average only by a factor of 1.03.

In our experiments we observed oscillations of small amplitude in time direction on fine leaves where the mesh is coarsened in time. These oscillations can be damped by adding an additional diffusion in time as discussed in Section 6.3.2. However, the strength of the stabilization term needs to be carefully adjusted since an overly strong diffusion reduces the speed of the depolarization front and negatively impacts the accuracy of the measured depolarization times. Stabilization can also significantly reduce the number of linear solver iterations required within the non-linear Newton solver.



(a) *Space-time mesh obtained with the standard maximum-based marking strategy.*

(b) *Space-time mesh obtained with the modified marking strategy.*

Figure 6.15. *Non-conforming adaptively refined space-time mesh on* $(-1,1) \times (3.75,7.75)$ *using a standard maximum-based refinement strategy (left) and weighted error indicators (right). The vertical axis equals the time.*

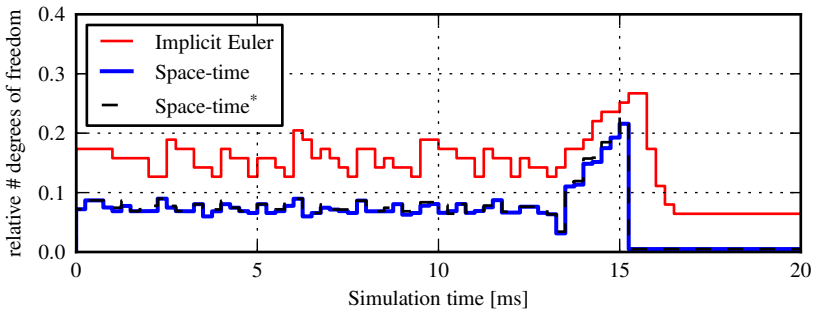


Figure 6.16. *Number of degrees of freedom relative to the dimension 33,345 of a conforming ansatz space on a uniform mesh. For the implicit Euler method, accumulated number of degrees of freedom are shown. For the space-time discretization results with standard and modified marking strategy are shown.*

6.3.6 (2+1)-dimensional Monodomain Equation

We consider the solution of the $(2+1)$-dimensional monodomain equation on the domain $\Omega = (0,1)^2$. For the following experiments we employed a two-dimensional version of the conductivity tensor used for the three-dimensional small-scale problem in Sections 4.8.2 and 5.5.1, i.e.,

$$\mathbf{G}_{\text{mono}} = 2 \cdot \mathbf{a}_\text{l} \otimes \mathbf{a}_\text{l} + 0.25562 \cdot (\mathbf{a}_\text{t} \otimes \mathbf{a}_\text{t} + \mathbf{a}_\text{n} \otimes \mathbf{a}_\text{n}) \ \text{mS/cm} \ .$$

with

$$\mathbf{a}_\text{l} = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]^\text{T}, \ \mathbf{a}_\text{t} = \mathbf{0}, \ \mathbf{a}_\text{n} = \left[\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right]^\text{T} \ .$$

We applied a stimulation current of $I_\text{app} = 250 \ \mu\text{A/cm}^2$ for $\frac{1}{4}$ ms in $(\frac{3}{8}, \frac{5}{8})^2 \subset \Omega$. For our experiments we used space-time slabs of extent $\frac{1}{8}$ ms in time direction. The coarse tessellation consisted of $8 \times 8 \times 1$ hexahedra. The maximal tree depth was set to $\ell_\text{max} = 3$ and $(\delta_\ell)_{\ell=1}^3 = (1/4, 1/4, 1/4)$. We used a conforming ansatz space $\mathbb{Y}^\text{c}_{\mathcal{T}}$. Refinement was driven by the strategy used in Section 6.3.5. We employed a Newton solver with backtracking as in the previous section. The arising linear systems were solved with a GMRES(30) iterative solver, preconditioned by a one-level restricted additive Schwarz (RAS) preconditioner with an overlap of four. The problem was stabilized by the diffusion term

$$\sum_{E \in \mathcal{T}_\tau} 10^{-2} \operatorname{diam}(E) \ (\partial_t V, \partial_t U)_{L^2(E)} \ . \tag{6.22}$$

Stabilization was used primarily to ensure convergence of the linear solver which failed repeatedly to reach the (absolute and relative) tolerance of 10^{-8} and thus slowed down the Newton solver when no stabilization was used.

In Figure 6.17 the number of Newton solver iterations and function evaluations over time are shown. In most laps the Newton converges within at 5–6 iterations and requires 11–14 function evaluations. The RAS preconditioned GMRES solver converges within at most 106 iterations. Note

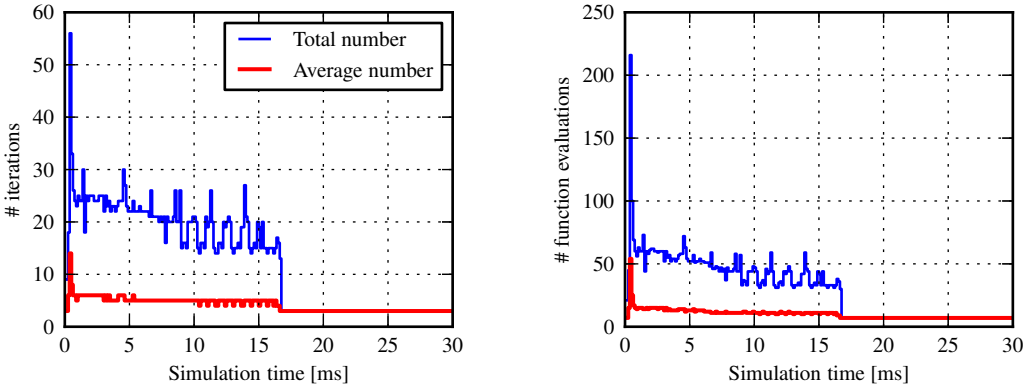


Figure 6.17. *Number of Newton iterations (left) and functional evaluations (right). The plot shows the number of iterations and evaluations accumulated (blue) and averaged (red) over all passes.*

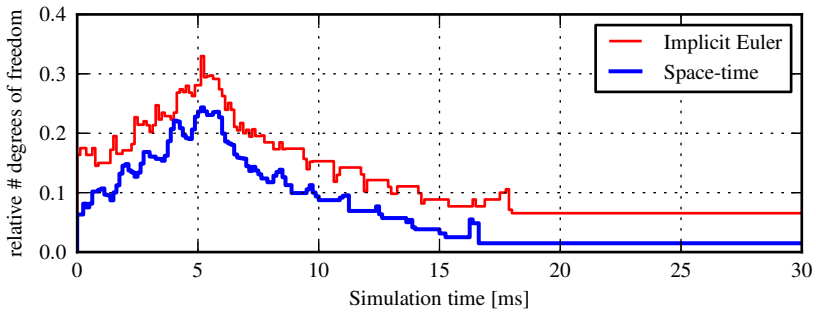


Figure 6.18. *Number of degrees of freedom relative to the dimension 282,897 of a conforming ansatz space on a uniform mesh.*

that the stabilization term is crucial to ensure convergence of the linear solver despite the large overlap used in this experiment.

Figure 6.18 shows the reduction in the number of degrees of freedom relative to a conforming discretization with the same minimal mesh width. Due to the use of a finer tessellation and space-time slabs with smaller extent in the time direction we measure a lower reduction compared to the results from Section 6.3.5. An average improvement by a factor of 1.63 and 2.99 for the first 15 ms and 30 ms simulation time, respectively, is measured over a spatially adaptive method of lines discretization with $\tau = 0.0078125$ and $L_{\text{lap}} = 16$.

Figure 6.19 shows selected contour surfaces of the membrane voltage V_τ on the first 96 space-time slabs together with the mesh on levels ≥ 2. We refer to Figure 2.3 and Figure 5.4 for different visualizations of the membrane voltage solution for the same problem computed by means of an implicit-explicit Euler discretization.

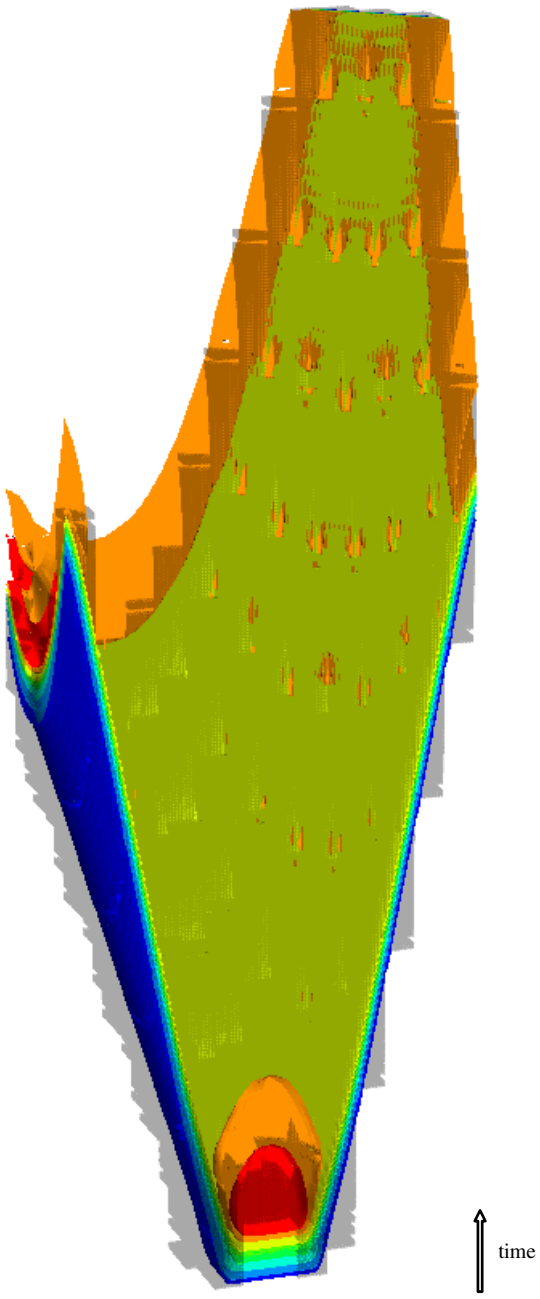


Figure 6.19. *Space-time contour plot of the membrane voltage on* $(0,1) \times \left(\frac{1}{2},1\right) \times (0,12)$ *computed using space-time finite elements. The wireframe of the mesh on leaves with level* ≥ 2 *is overlayed to indicate the structure of the adaptively refined meshes. The time direction is scaled by a factor* $\frac{1}{4}$ *for the visualization.*

6.4 Related Work

Local time stepping is a standard technique for the solution of hyperbolic equations using explicit time discretization schemes in combination with block-structured adaptive mesh refinement methods[21,22,56]. Gassner et al.[69] discuss local time stepping for discontinuous Galerkin discretizations based on explicit predictor-corrector schemes. Local time stepping in the context of wavelet-based adaptive resolution schemes using explicit time discretization schemes is discussed, for example, by Domingues et al.[57] and Bendahmane et al.[19,20]. Coquel et al.[48] discuss local time stepping for semi-implicit discretization schemes and adaptive resolution methods.

For a historical review of locally adaptive time stepping techniques we refer to Gander and Halpern[67].

Griebel and Oeltz[71] use space-time sparse grids for the solution of parabolic partial differential equations with continuous and discontinuous ansatz functions in time. By using sparse grids, the dimension of the ansatz space can be reduced from $O\left(N^{d+1}\right)$ to $O\left(N^{d}\right)$, i.e., the order of the dimension of a stationary problem[72]. In contrast to classical sparse grids, the space-time sparse grids constructed by Griebel and Oeltz are not limited to tensor product spaces but can be used with an arbitrary multi-level basis in space. The authors discuss an adaptive discretization of parabolic equations with non-smooth solutions for which the regularity requirements of the sparse-grid approximation does not hold.

Yu[176] describes an implementation of local time stepping based on a multiplicative Schwarz domain decomposition method. In this method the finite element space is decomposed according to an overlapping decomposition of the domain Ω. Using the method of lines, the considered time-dependent partial differential is reduced to a coupled set of ordinary differential equations in the finite element space. These equations are solved in the interior of the local subdomains and the solutions in the subspaces are combined using a multiplicative Schwarz algorithm. Since the ordinary differential equations in different subdomains are solved independently (though in sequential order), different time steps or even different discretization schemes may be used in different domains. This approach has later been combined with block-structured AMR[177].

Tezduyar and Sathe[152] propose the *enhanced-discretization space-time technique* (EDSTT) to enable local time stepping for fluid dynamics and fluid structure interaction. They introduce two techniques. The first method, EDSTT-SM (single mesh), is based on a space-time conforming mesh that is refined towards the region of interest. The second method, EDSTT-MM (multi mesh), uses overlapping meshes of different resolution. In this method, the solution is locally written as a sum of contributions from ansatz spaces corresponding to the different meshes[153]. The authors present $(1+1)$- and $(2+1)$-dimensional results. Similar to this work, we propose to use a space-time discretization to implement local time stepping in an implicit setting. Both works are based on a discontinuous Galerkin approximation in time. In contrast to Tezduyar and Sathe we use non-overlapping, non-conforming meshes (as discussed in Chapter 5) and standard finite element ansatz spaces for the discretization within a space-time slab. This ensures that our method can be used for solving a large class of partial differential equations. Whereas Tezduyar and Sathe consider a fixed

spatial region where a smaller time step is used, in our approach the time step is adapted based on error estimates.

Weinzierl and Köppl [164] study a space-time multi-grid method for the solution of the heat equation on adaptive tree-based meshes. The authors use a stencil-based implementation of a finite element discretization and discuss a full space-time multi-grid algorithm. Results for the adaptive solution of the $(2+1)$-dimensional heat equation are reported.

Neumüller and Steinbach [117] describe a space-time discontinuous Galerkin method for the solution of the transient heat equation. The authors discuss the refinement of pentatopes in $\mathbb{R}^4$ for the purpose of generating space-time adaptive meshes. Convergence studies for the $(3+1)$-dimensional heat equation and a solution of the $(2+1)$-dimensional Navier Stokes equation on a moving domain are presented. An example of a space-time adaptive solution of the $(1+1)$-dimensional heat equation is given. In contrast to Neumüller and Steinbach we use the discontinuous Galerkin discretization in combination with non-conforming finite elements. By using cubes instead of tetrahedra or pentatopes, the handling of arbitrary dimensions is straightforward.

6.5 Discussion

We have proposed the use of space-time discretizations to enable local time stepping for the solution of time-dependent partial differential equations for which global time step control is inefficient. By combining finite element ansatz spaces built on non-conforming meshes with a discontinuous Galerkin discretization in time, we can reuse existing adaptive techniques (see Chapter 4 and Chapter 5) while having the possibility to control the computational and storage requirements by means of the extent of the space-time slabs. In order to simplify the assembly of the jump term, we use a space-time transfer operator to map the trace of the solution on the previous space-time slab to the new slab. Let us point out that, in particular for $(3+1)$-dimensional simulations, lightweight data structures are a crucial ingredient for large-scale simulations due to the limited local memory available on current and next-generation supercomputers.

In Sections 6.3.1–6.3.4 we have studied the solution of the heat equation for different spatial dimensions. We have measured a consistent gain by a factor of approximately two in the reduction of the number of degrees of freedom when comparing our space-time adaptive discretization with a spatially adaptive method of lines discretization. The best of our knowledge, space-time adaptivity for $(3+1)$-dimensional partial differential equations has not been demonstrated before.

In Section 6.3.2 we observed instabilities when using a non-conforming mortar ansatz space instead of a conforming finite element space and discussed the stabilization of the method. Our experimental results indicate that the large kernel of the space-time discretization of the Laplacian is a major source of complications for both the construction of stable discretizations and the efficient preconditioning of the arising linear systems.

We have demonstrated the feasibility of the adaptive space-time discretization of reaction-diffusion equations, in particular of the monodomain equation. The arising large non-linear systems of equations can be solved with a globalized Newton method. The efficient preconditioning of the

arising linear systems is however challenging (see below).

In order to take advantage of space-time adaptivity, robust and efficient preconditioning techniques are crucial. Several authors have studied space-time multi-grid algorithms[82,83] and multi-grid wavefront relaxation methods[93] for the solution of transient linear parabolic partial differential equations. Weinzierl and Köppl[164] demonstrated a space-time multi-grid on adaptive meshes. Due to the difference in the discretizations these results cannot be applied straightforwardly to our setup.

Despite the significant reduction in the number of degrees of freedom through the use of space-time adaptivity that we observed in our experiments, it is not clear if it is possible to obtain an equally large reduction in the computing time. In fact, even with linear solvers of optimal complexity, computing time will increase with the dimension due to the growing matrix bandwidth. More generally, we note that the curse of dimension not only shows in the exponential growth of the total number of degrees of freedom but is also reflected "locally". For example, the number of local degrees of freedom (and thus the matrix bandwidth) and number of quadrature points also grow exponentially with the dimension. Thus, a space-time solver needs to perform less work per entry in the stiffness matrix in order to be competitive to a series of lower-dimensional solvers.

Let us point out that a space-time discretization can be an attractive alternative to a method of lines discretization for other reasons. Space-time discretizations allow for a transparent treatment of moving domains. In the context of computational electrocardiology this is of interest for the simulation of coupled electromechanical models. Additionally, space-time adaptive discretization techniques open up an additional level of parallelism in the workload since the serial time stepping is replaced by a global space-time iterative procedure. In fact, in all our parallel runs the space-time slabs were distributed according to a $(d+1)$-dimensional space-filling curve. For this reason, space-time discretizations may be of interest for a new generation of algorithms targeted at exa-scale supercomputers. Since, however, memory bandwidth will be a scarce resource in such systems it is not clear if the additional parallelism can be taken advantage of in practice due to the "local curse of dimension" (see above). Nevertheless, space-time discretization should be investigated as an alternative to parallel-in-time integration schemes (see, for example, Speck et al.[146]) which usually feature strict bounds on the achievable speedup.

7 Conclusion

We presented adaptive discretization schemes for the solution of reaction-diffusion equations in the field of computational electrocardiology. The presented methods combine the plainness of structured meshes with the flexibility of a non-conforming mortar element discretization in a novel and original way.

We presented two adaptive mesh data structures that use either a conforming tessellation or a forest of shallow trees to organize the local structured meshes. The first method allows for a representation of the complete mesh by a single integer vector $\boldsymbol{\ell} \in \mathbb{Z}_{\geq 1}^N$. This data structure is not only extremely lightweight but also lacks the implicit storage of a refinement history and thus can be modified very easily. A disadvantage of this mesh data structure is the limited reduction in the degrees of freedom that is measured in practice. The second data structure is based on shallow trees, i.e., the representation of the mesh by a vector $\boldsymbol{\tau} \in \left((\mathbb{Z}_{\geq 0})^*\right)^N$ of 2^d-trees. This data structure has a higher memory footprint and is less flexible with respect to mesh modifications but, due to the hierarchical structure, provides more control over the location and shape of the refined region.

We described two approaches for the solution of variational problems on these mesh data structures. First we proposed and evaluated a matrix-free scheme for the monodomain equation with a tailored block preconditioner. This approach allows for exploiting the special structure of the meshes $\mathscr{T}_{\boldsymbol{\ell}}$ or $\mathscr{T}_{\boldsymbol{\tau}}$ but is not well suited for ill-conditioned problems. As an alternative we described the construction of standard linear algebra data structures on non-conforming meshes. In particular we discussed the element-wise assembly of stiffness matrices on subspaces of the product space via an algebraic representation of the inclusion map.

The presented results do confirm our initial research hypothesis, namely that by embracing non-conformity, a rich flavor of adaptive strategies is at our disposal. These methods can be tailored to different design goals such as the simplicity of the implementation (Chapter 4) or flexibility (Chapter 5).

The presented schemes can be used in the context of spatial adaptivity (possibly in combination with global time stepping) or space-time adaptivity using a space-time discretization. Due to the special structure of the space-time discrete problem, stabilization might be necessary. We have presented extensive numerical experiments, including the space-time adaptive solution of the $(3+1)$-dimensional heat equation and the $(2+1)$-dimensional monodomain equation with a realistic Bernus membrane model, that prove the feasibility of this idea.

In this work we considered the development of adaptive schemes (i.e., the combination of mesh and algebra data structures, linear solver and marking strategy) as a design process that requires selection between potentially competing design choices. Therefore one expects that multiple iterations are required to obtain an (Pareto-) optimal method. An additional complication stems from the interaction between these individual components. For example, our results clearly show that efficient preconditioners for uniform mesh methods may be less appropriate for the considered adaptive meshes. Moreover, the choice of the mesh data structure has a profound impact on the distribution of the accumulated error indicators and thus on the effectiveness of the marking strategy. Therefore it is not unexpected that much work is left for the future. Here, we want to point out several topics that should be the subject of future research.

Algebra data structures and preconditioners. The construction of efficient preconditioning techniques is an important topic for future work. On the one hand, efficient preconditioners for the non-conforming discretization of elliptic problems are required (see, for example, Section 5.6). On the other hand, preconditioners for the space-time discretization of reaction-diffusion equations need to be developed (see Section 6.5). For the discretization of elliptic problems using a mortar element discretization, several theoretical studies already exist that can serve as a starting point (see Section 4.10).

Our results show that it can be advantageous to "decouple" the choice of the ansatz space from the choice of the basis (of a superspace) used to represent the solution. In doing so, one can optimize the ansatz space with respect to stability and approximation properties while choosing an appropriate data representation for good performance. The design of good preconditioning techniques which can exploit these data structures is crucial. In Chapter 4 we used a product space representation but other vector spaces may be employed. A particular advantage of the product space representation was the block structure of the stiffness matrix which allowed us to perform local re-assembly of the stiffness matrix.

Error indication and marking strategies. In the numerical experiments in this thesis we have used a maximum-based marking strategy together with accumulated residual-based error estimators or gradient error indicators. The sum of all error indicators/estimators is not a suitable measure for the optimality of a mesh when restricting the maximal refinement level. Therefore we used the termination of the refinement process as an indicator for the optimality of the constructed mesh. An obvious disadvantage of this approach is a relatively high number of required repetitions in particular when using deeper trees (see Section 5.5).

Since accumulation alters the distribution of the error indicators we believe that tailored marking strategies can give a significant reduction in the number of passes required to find a suitable mesh. Moreover, a strategy is required to assess the quality or optimality of the adaptive meshes. This necessitates further verification of our methods on different geometries using different error indication and marking strategies. The goal is to deliver a robust scheme that minimizes the need for manual parameter tuning by domain scientists.

Emerging architectures. The presented schemes have been implemented and evaluated on homo-

geneous parallel computers. Since accelerators in the form of graphics processing elements or other throughput-oriented co-processors are increasingly common in high-end supercomputers, the adaptation of the presented adaptive schemes to such hybrid architectures should be the subject of future work. The structured meshes serving as the fundamental building block of our adaptive meshes are well suited for throughput-optimized chips, too. However, in contrast to central processing units, we expect the size of the local meshes to have a higher impact on the sustained performance.

The novel combination of non-conforming discretizations with optimized lightweight data structures as presented in this thesis provides us with an exciting new class of adaptive methods. Our results clearly demonstrate the viability of this approach and encourages further research in the areas mentioned above.

A Assembly of the Mortar Projection

In this appendix we provide a detailed description of the assembly of the mortar projection, in particular the matrix $\mathbf{R}$ as defined in equation (4.10b). As mentioned in Section 4.3.3 we need to take into account the different orientations of the structured meshes on the slave and master sides induced by the orientation of the adjacent patches.

To simplify the notation we restrict ourselves to the case $d = 3$. We consider a non-mortar $\gamma_m^+ = \bigcup_{m'} \gamma_{m'}^-$ on the interface Γ_{ij}. In the geometrically conforming setting in Chapter 4 we have $\gamma_m^+ = \gamma_m^- = \Gamma_{ij}$. In the geometrically non-conforming setting discussed in Chapter 5 the non-mortar in general is split into multiple mortars. Moreover γ_m^+ and Γ_{ij} usually are not equal. Without loss of generality we assume that the non-mortar side is associated with the patch Ω_j.

For an axis-aligned rectangles $X \subseteq (0,1)^2$ we denote by S_X the unique affine linear bijection $(0,1)^2 \to X$. As in Section 4.3.3 we denote the parametrizations of γ_m^+ and $\gamma_{m'}^-$ over $(0,1)^2$ by φ_m^+ and $\varphi_{m'}^-$. By φ_i, φ_j we denote the parametrizations of the patches Ω_i and Ω_j over $(0,1)^3$. Furthermore we define

$$\widetilde{\varphi_i} : (0,1)^2 \cong \varphi_i^{-1}(\Gamma_{ij}) \xrightarrow{\varphi_i} \Gamma_{ij} \quad \text{and} \quad \widetilde{\varphi_j} : (0,1)^2 \cong \varphi_j^{-1}(\Gamma_{ij}) \xrightarrow{\varphi_j} \Gamma_{ij} .$$

By design of the mesh data structures $\mathscr{T}_{\boldsymbol{\ell}}$ and $\mathscr{T}_{\boldsymbol{\tau}}$ the parametrizations φ_m^+ and $\varphi_{m'}^-$ are induced by the parametrizations of the adjacent patches, i.e.,

$$\varphi_m^+ = \widetilde{\varphi_j} \circ S_{\left(\varphi_m^+\right)^{-1}\left(\gamma_m^+\right)} \quad \text{and} \quad \varphi_{m'}^- = \widetilde{\varphi_i} \circ S_{\left(\varphi_{m'}^-\right)^{-1}\left(\gamma_{m'}^-\right)} . \tag{A.1}$$

The function $A = \left(\widetilde{\varphi_j}\right)^{-1} \circ \widetilde{\varphi_i}$ maps corners of $(0,1)^2$ to corners. One can show that there exists a vector $\mathbf{b} \in \{0,1\}^2$ and a permutation $\omega \in \mathbb{S}_2$ such that

$$A(\mathbf{x}) = \mathbf{b} - (\mathbf{1} - 2\operatorname{diag}\mathbf{b})\,\mathbf{E}_\omega \mathbf{x}$$

with $(\mathbf{E}_\omega)_{kh} = \delta_{k\omega(h)}$.

According to definition (4.10b) we have

$$\mathbf{R}_{\hat{\alpha}\varepsilon} = \int_{\gamma_m^+} \psi_{\hat{\alpha}} \theta_\varepsilon \, \mathrm{d}S(\mathbf{x}) = \sum_{m'} \int_{\gamma_{m'}^-} \psi_{\hat{\alpha}} \theta_\varepsilon \, \mathrm{d}S(\mathbf{x}) = \sum_{m'} \sum_{F^- \subset \gamma_{m'}^-} \int_{F^-} \psi_{\hat{\alpha}} \theta_\varepsilon \, \mathrm{d}S(\mathbf{x}) . \tag{A.2}$$

Since we employ locally nested meshes in Chapters 4 and 5 for each face $F^- \subset \gamma_{m'}^-$ there exists a unique face $F^+ \subset \gamma_m^+$ with $F^- \subseteq F^+$. By $\widehat{F^-} \subseteq (0,1)^2$ and $\widehat{F^+} \subseteq (0,1)^2$ we denote the images of these faces under the inverse parametrizations $\widetilde{\varphi}_i$ and $\widetilde{\varphi}_j$, respectively. We use the short-hand notations

$$\varphi_{F^-} = \widetilde{\varphi}_i \circ S_{\widehat{F^-}} \quad \text{and} \quad \varphi_{F^+} = \widetilde{\varphi}_j \circ S_{\widehat{F^+}} \,.$$

Let us point out that is convenient to work with the parametrizations of the interface Γ_{ij} rather than with the parametrizations of mortars and non-mortars in a geometrically non-conforming setting.

By definition (4.12)

$$\psi_{\dot{\alpha}}\big|_{F^+} = \frac{w_{\dot{\alpha}}}{\sqrt{\det\left((\nabla\varphi_m^+)^{\mathrm{T}}\nabla\varphi_m^+\right)}}\hat{\psi}_{\dot{\alpha}} \circ (\varphi_{F^+})^{-1}$$

and $\theta_\varepsilon\big|_{F^-} = \hat{\theta}_\varepsilon \circ (\varphi_{F^-})^{-1}$.

In order to evaluate the surface integral over F^- in equation (A.2) we use the parametrization

$$\xi = \widetilde{\varphi}_j \circ S_{A\left(\widehat{F^-}\right)} = \widetilde{\varphi}_i \circ A^{-1} \circ S_{A\left(\widehat{F^-}\right)} \,.$$

With this definition we have

$$\begin{aligned}
\psi_{\dot{\alpha}} \circ \xi &= \frac{w_{\dot{\alpha}}}{\sqrt{\det\left((\nabla\varphi_m^+)^{\mathrm{T}}\nabla\varphi_m^+\right)}}\hat{\psi}_{\dot{\alpha}} \circ (\varphi_{F^+})^{-1} \circ \widetilde{\varphi}_j \circ S_{A\left(\widehat{F^-}\right)} \\
&= \frac{w_{\dot{\alpha}}}{\sqrt{\det\left((\nabla\varphi_m^+)^{\mathrm{T}}\nabla\varphi_m^+\right)}}\hat{\psi}_{\dot{\alpha}} \circ S_{\widehat{F^+}}^{-1} \circ S_{A\left(\widehat{F^-}\right)}
\end{aligned}$$

and

$$\begin{aligned}
\theta_\varepsilon \circ \xi &= \hat{\theta}_\varepsilon \circ (\varphi_{F^-})^{-1} \circ \widetilde{\varphi}_i \circ A^{-1} \circ S_{A\left(\widehat{F^-}\right)} \\
&= \hat{\theta}_\varepsilon \circ S_{\widehat{F^-}} \circ A^{-1} \circ S_{A\left(\widehat{F^-}\right)} \\
&= \hat{\theta}_\varepsilon \circ A^{-1} \,.
\end{aligned}$$

The last equality is a consequence of the identity $A \circ S_X = S_{A(X)} \circ A$ which itself follows from geometric considerations. If $\sigma \in \mathbb{S}_4$ is the permutation of the corners of $(0,1)^2$ induced by the mapping A we have $\hat{\theta}_\varepsilon \circ A^{-1} = \hat{\theta}_{\sigma(\varepsilon)}$.

Using the parametrization ξ and the equations above we finally obtain

$$\mathbf{R}_{\dot{\alpha}\varepsilon} = \sum_{m'} \sum_{F^- \subset \gamma_{m'}^-} w_{\dot{\alpha}} \int_{(0,1)^2} \left(\hat{\psi}_{\dot{\alpha}} \circ S_{\widehat{F^+}}^{-1} \circ S_{A\left(\widehat{F^-}\right)}\right) \hat{\theta}_{\sigma(\varepsilon)} \frac{\sqrt{\det\left((\nabla\xi)^{\mathrm{T}}\nabla\xi\right)}}{\sqrt{\det\left((\nabla\varphi_m^+)^{\mathrm{T}}\nabla\varphi_m^+\right)}} \,\mathrm{d}\mathbf{x} \,. \tag{A.3}$$

The integral in equation (A.3) can be approximated with a standard Gauss-Legendre quadrature rule.

Bibliography

1. G. S. Abdoulaev, Y. Achdou, Y. A. Kuznetsov, and C. Prud'homme. On a parallel implementation of the mortar element method. *ESAIM-Math. Model. Num.*, 33(2):245–259, 1999.

2. R. Abedi, B. Petracovici, and R. B. Haber. A space-time discontinuous Galerkin method for linearized elastodynamics with element-wise momentum balance. *Comput. Meth. Appl. Mech. Eng.*, 195(25–28):3247–3273, 2006.

3. P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. M.*, 184(2–4):501–520, 2000.

4. P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix. Anal. A*, 23(1):15–41, 2001.

5. M. Anderson and J.-H. Kimn. A numerical approach to space-time finite elements for the wave equation. *J. Comput. Phys.*, 226(1):466–476, 2007.

6. A. K. Aziz and P. Monk. Continuous finite elements in space and time for the heat equation. *Math. Comput.*, 52(186):255–274, 1989.

7. I. Babuska and M. Suri. The *p* and *h-p* versions of the finite element method, basic principles and properties. *SIAM Rev.*, 36(4):578–632, 1994.

8. M. Bader, S. Schraufstetter, C. A. Vigh, and J. Behrens. Memory efficient adaptive mesh generation and implementation of multigrid algorithms using Sierpinski curves. *Int. J. Comput. Sci. Eng.*, 4(1):12–21, 2008.

9. M. Bader, C. Böck, J. Schwaiger, and C. Vigh. Dynamically adaptive simulations with minimal memory requirement – Solving the shallow water equations using Sierpinski curves. *SIAM J. Sci. Comput.*, 32(1):212–228, 2010.

10. M. Bader, K. Rahnema, and C. Vigh. Memory-efficient Sierpinski-order traversals on dynamically adaptive, recursively structured triangular grids. In K. Jónasson, editor, *Applied Parallel and Scientific Computing*, volume 7134 of *Lecture Notes in Computer Science*, pages 302–312. Springer, 2012.

11. S. Balay, W. D. Gropp, L. Curfman McInnes, and B. 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.

12. S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang. PETSc web page, 2011. http://www.mcs.anl.gov/petsc.

13. W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Trans. Math. Softw.*, 38(2): 14:1–14:28, 2012.

14. P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuss, H. Rentz-Reichert, and C. Wieners. UG – A flexible software toolbox for solving partial differential equations. *Comput. Vis. Sci.*, 1(1):27–40, 1997.

15. G. W. Beeler and H. Reuter. Reconstruction of the action potential of ventricular myocardial fibres. *J. Physio.*, 268(1):177–210, 1977.

16. F. B. Belgacem. The mortar finite element method with Lagrange multipliers. *Numer. Math.*, 84(2):173–197, 1999.

17. Y. Belhamadia. A time-dependent adaptive remeshing for electrical waves of the heart. *IEEE Trans. Biomed. Eng.*, 55(2):443–452, 2008.

18. Y. Belhamadia, A. Fortin, and Y. Bourgault. Towards accurate numerical method for monodomain models using a realistic heart geometry. *Math. Biosci.*, 220(2):89–101, 2009.

19. M. Bendahmane, R. Bürger, R. Ruiz-Baier, and K. Schneider. Adaptive multiresolution schemes with local time stepping for two-dimensional degenerate reaction–diffusion systems. *Appl. Numer. Math.*, 59(7):1668–1692, 2009.

20. M. Bendahmane, R. Bürger, and R. Ruiz-Baier. A multiresolution space-time adaptive scheme for the bidomain model in electrocardiology. *Numer. Meth. Part. D. E.*, 26(6):1377–1404, 2010.

21. M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82(1):64–84, 1989.

22. M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53(3):484–512, 1984.

23. C. Bernardi and F. Hecht. Error indicators for the mortar finite element discretization of the laplace equation. *Math. Comput.*, 71(240):1371–1403, 2002.

24. C. Bernardi and Y. Maday. Mesh adaptivity in finite elements using the mortar method. *Revue Européenne des Éléments*, 9(4):451–465, 2000.

25. C. Bernardi, Y. Maday, and A. T. Patera. A new nonconforming approach to domain decomposition: The mortar element method. In H. Brezis and J. L. Lions, editors, *Nonlinear Partial Differential Equations and Their Applications*, 299, pages 13–51. Pitman Res. Notes Math. Ser., 1994.

26. C. Bernardi, Y. Maday, and F. Rapetti. Basics and some applications of the mortar element method. *GAMM-Mitt.*, 28(2):97–123, 2005.

27. O. Bernus, R. Wilders, C. W. Zemlin, H. Verschelde, and A. V. Panfilov. A computationally efficient electrophysiological model of human ventricular cells. *Am. J. Physiol.-Heart C.*, 282 (6):H2296–H2308, 2002.

28. M. J. Bishop, G. Plank, R. A. B. Burton, J. E. Schneider, D. J. Gavaghan, V. Grau, and P. Kohl. Development of an anatomically detailed MRI-derived rabbit ventricular model and assessment of its impact on simulations of electrophysiological function. *Am. J. Physiol.-Heart C.*, 298(2):H699–H718, 2010.

29. P. E. Bjørstad, M. Dryja, and T. Rahman. Additive Schwarz methods for elliptic mortar finite element problems. *Numer. Math.*, 95(3):427–457, 2003.

30. P. B. Bochev, M. D. Gunzburger, and J. N. Shadid. Stability of the SUPG finite element method for transient advection-diffusion problems. *Comput. Meth. Appl. Mech. Eng.*, 193 (23–26):2301–2323, 2004.

31. R. Bordas, B. Carpentieri, G. Fotia, F. Maggio, R. Nobes, J. Pitt-Francis, and J. Southern. Simulation of cardiac electrophysiology on next-generation high-performance computers. *Philos. T. R. Soc. A*, 367(1895):1951–1969, 2009.

32. D. Braess, W. Dahmen, and C. Wieners. A multigrid algorithm for the mortar finite element method. *SIAM J. Numer. Anal.*, 37(1):48–69, 2000.

33. A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Math. Comput.*, 31 (138):333–390, 1977.

34. A. N. Brooks and T. J. R. Hughes. Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. *Comput. Meth. Appl. Mech. Eng.*, 32(1–3):199–259, 1982.

35. C. Burstedde, O. Ghattas, M. Gurnis, G. Stadler, E. Tan, T. Tu, L. C. Wilcox, and S. Zhong. Scalable adaptive mantle convection simulation on petascale supercomputers. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 621–6215, 2008.

36. C. Burstedde, O. Ghattas, G. Stadler, T. Tu, and L. C. Wilcox. Towards adaptive mesh PDE simulations on petascale computers. In *Proceedings of TeraGrid 2008*, 2008.

37. C. Burstedde, O. Ghattas, M. Gurnis, T. Isaac, G. Stadler, Tim Warburton, and L. C. Wilcox. Extreme-scale AMR. In *Proceedings of the 2010 ACM/IEEE conference on Supercomputing*, pages 1–12, 2010.

38. C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.*, 33(3):1103–1133, 2011.

39. H. C. Chen and T. S. Huang. A survey of construction and manipulation of octrees. *Comput. Vis. Graphics & Imag. Proc.*, 43(3):409–431, 1988.

40. E. M. Cherry, H. S. Greenside, and C. S. Henriquez. A space-time adaptive method for simulating complex cardiac dynamics. *Phys. Rev. Lett.*, 84(6):1343–1346, 2000.

41. E. M. Cherry, H. S. Greenside, and C. S. Henriquez. Efficient simulation of three-dimensional anisotropic cardiac tissue using an adaptive mesh refinement method. *Chaos*, 13(3):853–865, 2003.

42. R. H. Clayton, O. Bernus, E. M. Cherry, H. Dierckx, F. H. Fenton, L. Mirabella, A. V. Panfilov, F. B. Sachse, G. Seemann, and H. Zhang. Models of cardiac tissue electrophysiology: Progress, challenges and open questions. *Prog. Biophys. Mol. Bio.*, 104(1–3):22–48, 2011.

43. B. Cockburn. Discontinuous Galerkin methods. *ZAMM-Z. Angew. Math. ME*, 83(11): 731–754, 2003.

44. A. Cohen, S. M. Kaber, S. Müller, and M. Postel. Fully adaptive multiresolution finite volume schemes for conservation laws. *Math. Comput.*, 72(241):183–225, 2003.

45. P. Colella, D. T. Graves, N. D. Keen, T. J. Ligocki, D. F. Martin, P. W. McCorquodale, D. Modiano, P. O. Schwartz, T. D. Sternberg, and B. Van Straalen. *Chombo Software Package for AMR Applications: Design Document*, 2009.

46. P. Colli Franzone and L. F. Pavarino. A parallel solver for reaction-diffusion systems in computational electrocardiology. *Math. Model Meth. App. Sci.*, 14(06):883–911, 2004.

47. P. Colli Franzone, P. Deuflhard, B. Erdmann, J. Lang, and L. F. Pavarino. Adaptivity in space and time for reaction-diffusion systems in electrocardiology. *SIAM J. Sci. Comput.*, 28(3): 942–962, 2006.

48. F. Coquel, Q. L. Nguyen, M. Postel, and Q. H. Tran. Local time stepping applied to implicit-explicit methods for hyperbolic systems. *SIAM Mult. Mod. Simul.*, 8(2):540–570, 2010.

49. T. A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):165–195, 2004.

50. T. A. Davis. Algorithm 832: UMFPACK v4.3 – an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):196–199, 2004.

51. T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix. Anal. A*, 18(1):140–158, 1997.

52. T. A. Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. Math. Softw.*, 25(1):1–20, 1999.

53. H. De Sterck, U. M. Yang, and J. J. Heys. Reducing complexity in parallel algebraic multigrid preconditioners. *SIAM J. Matrix. Anal. A*, 27(4):1019–1039, 2006.

54. M. Delfour, W. Hager, and F. Trochu. Discontinuous Galerkin methods for ordinary differential equations. *Math. Comput.*, 36(154):455–473, 1981.

55. P. Deuflhard, B. Erdmann, R. Roitzsch, and G. T. Lines. Adaptive finite element simulation of ventricular fibrillation dynamics. *Comput. Vis. Sci.*, 12(5):201–205, 2009.

56. L. F. Diachin, R. Hornung, P. Plassmann, and A. Wissink. Parallel adaptive mesh refinement. In M. A. Heroux, P. Raghavan, and H. D. Simon, editors, *Parallel Processing for Scientific Computing*, chapter 8, pages 143–163. SIAM, 2006.

57. M. O. Domingues, S. M. Gomes, O. Roussel, and K. Schneider. An adaptive multiresolution scheme with local time stepping for evolutionary PDEs. *J. Comput. Phys.*, 227(8):3758–3780, 2008.

58. K. Eriksson, C. Johnson, and V. Thomée. Time discretization of parabolic problems by the discontinuous Galerkin method. *ESAIM-Math. Model. Num.*, 19(4):611–643, 1985.

59. A. Ern and J.-L. Guermond. *Theory and Practice of Finite Elements*. Applied Mathematical Sciences. Springer, 2004.

60. M. Ethier and Y. Bourgault. Semi-implicit time-discretization schemes for the bidomain model. *SIAM J. Numer. Anal.*, 46(5):2443–2468, 2008.

61. S. Ethier, W. M. Tang, and Z. Lin. Gyrokinetic particle-in-cell simulations of plasma microturbulence on advanced computing platforms. *J. Phys. Conf. Ser.*, 16(1):1–15, 2005.

62. R. Falgout, A. Baker, V. E. Henson, U. M. Yang, T. Kolev, B. Lee, J. Painter, C. Tong, and P. Vassilevski. Hypre web page, 2011. https://computation.llnl.gov/casc/linear_solvers.

63. H. Feng, C. Mavriplis, R. Feng, and R. Biswas. Parallel 3D mortar element method for adaptive nonconforming meshes. *J. Sci. Comput.*, 27(1–3):231–243, 2006.

64. B. Flemisch and B. I. Wohlmuth. Stable Lagrange multipliers for quadrilateral meshes of curved interfaces in 3D. *Comput. Meth. Appl. Mech. Eng.*, 196(8):1589–1602, 2007.

65. D. A. French and T. E. Peterson. A continuous space-time finite element method for the wave equation. *Math. Comput.*, 65(214):491–506, 1996.

66. V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2): 170–231, 1998.

67. M. J. Gander and L. Halpern. Techniques for locally adaptive timestepping developed over the last two decades. In *Domain Decomposition Methods*, 2012. to appear.

68. G. Gassner, F. Lörcher, and C.-D. Munz. A discontinuous Galerkin scheme based on a space-time expansion II. Viscous flow equations in multi dimensions. *J. Sci. Comput.*, 34 (3):260–286, 2008.

69. G. Gassner, M. Dumbser, F. Hindenlang, and C.-D. Munz. Explicit one-step time discretizations for discontinuous Galerkin and finite volume schemes based on local predictors. *J. Comput. Phys.*, 230(11):4232–4247, 2011.

70. S. Götschel, M. Weiser, and A. Schiela. Solving optimal control problems with the Kaskade 7 finite element toolbox. In A. Dedner, B. Flemisch, and R. Klöfkorn, editors, *Advances in DUNE*, pages 101–112. Springer, 2012.

71. M. Griebel and D. Oeltz. A sparse grid space-time discretization scheme for parabolic problems. *Computing*, 81(1):1–34, 2007.

72. M. Griebel, D. Oeltz, and P. Vassilevski. Space-time approximation with sparse grids. *SIAM J. Sci. Comput.*, 28(2):701–727, 2006.

73. D. M. Harrild and C. S. Henriquez. A computer model of normal conduction in the human atria. *Circ. Res.*, 87(7):e25–e36, 2000.

74. L. Hart and S. McCormick. Asynchronous multilevel adaptive methods for solving partial differential equations on multiprocessors: Basic ideas. *Parallel Comput.*, 12(2):131–144, 1989.

75. A. Harten. Multiresolution algorithms for the numerical solution of hyperbolic conservation laws. *Commun. Pur. Appl. Anal.*, 48(12):1305–1342, 1995.

76. A. Henderson. *The ParaView Guide: A Parallel Visualization Application*. Kitware Inc., 2005.

77. M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.

78. A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physio.*, 117(4):500–544, 1952.

79. M. G. Hoogendijk, M. Potse, A. C. Linnenbank, A. O. Verkerk, H. M. den Ruijter, S. C. M. van Amersfoorth, E. C. Klaver, L. Beekman, C. R. Bezzina, P. G. Postema, H. L. Tan, A. G.

Reimer, A. C. van der Wal, A. D. J. ten Harkel, M. Dalinghaus, A. Vinet, A. A. M. Wilde, J. M. T. de Bakker, and R. Coronel. Mechanism of right precordial ST-segment elevation in structural heart disease: Excitation failure by current-to-load mismatch. *Heart Rhythm*, 7(2): 238–248, 2010.

80. R. H. W. Hoppe, Y. Iliash, Y. Kuznetsov, Y. Vassilevski, and B. Wohlmuth. Analysis and parallel implementation of adaptive mortar finite element methods. *J. Numer. Math.*, 6(3): 223–248, 1998.

81. R. D. Hornung, A. M. Wissink, and S. R. Kohn. Managing complex data and geometry in parallel structured AMR applications. *Eng. Comput.*, 22(3–4):181–195, 2006.

82. G. Horton. The time-parallel multigrid method. *Commun. Appl. Numer. M.*, 8(9):585–595, 1992.

83. G. Horton and S. Vandewalle. A space-time multigrid method for parabolic partial differential equations. *SIAM J. Sci. Comput.*, 16(4):848–864, 1995.

84. W. Huang, L. Kamenski, and J. Lang. Adaptive finite elements with anisotropic meshes. In A. Cangiani, R. L. Davidchack, E. Georgoulis, A. N. Gorban, J. Levesley, and M. V. Tretyakov, editors, *Numerical Mathematics and Advanced Applications 2011*, pages 33–42. Springer, 2013.

85. T. J. R. Hughes and G. M. Hulbert. Space-time finite element methods for elastodynamics: formulations and error estimates. *Comput. Meth. Appl. Mech. Eng.*, 66(3):339–363, 1988.

86. G. M. Hulbert and T. J. R. Hughes. Space-time finite element methods for second-order hyperbolic equations. *Comput. Meth. Appl. Mech. Eng.*, 84(3):327–348, 1990.

87. J. D. Hunter. Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.*, 9(3):90–95, 2007.

88. J. Hutter and Alessandro Curioni. Dual-level parallelism for ab initio molecular dynamics: Reaching teraflop performance with the CPMD code. *Parallel Comput.*, 31(1):1–17, 2005.

89. R. Ierusalimschy, L. H. de Figueiredo, and W. Celes. *Lua 5.1 Reference Manual*. Lua.Org, 2006.

90. IPM. IPM Homepage, 2009. http://ipm-hpc.sourceforge.net/.

91. V. Iyer, R. Mazhari, and R. L. Winslow. A computational model of the human left-ventricular epicardial myocyte. *Biophys. J.*, 87(3):1507–1525, 2004.

92. P. Jamet. Galerkin-type approximations which are discontinuous in time for parabolic equations in a variable domain. *SIAM J. Numer. Anal.*, 15(5):912–928, 1978.

93. J. Janssen and S. Vandewalle. Multigrid waveform relaxation on spatial finite element meshes: The continuous-time case. *SIAM J. Numer. Anal.*, 33(2):456–474, 1996.

94. R. Kaeppeli, S. C. Whitehouse, S. Scheidegger, U.-L. Pen, and M. Liebendörfer. FISH: A three-dimensional parallel magnetohydrodynamics code for astrophysical applications. *Astrophys. J. Suppl. S.*, 195(2):20, 2011.

95. G. Karypis and V. Kumar. Parallel multilevel series k-way partitioning scheme for irregular graphs. *SIAM Rev.*, 41(2):278–300, 1999.

96. J. Keener and J. Sneyd. *Mathematical Physiology: I: Cellular Physiology*. Interdisciplinary Applied Mathematics. Springer, 2nd edition, 2008.

97. J. Keener and J. Sneyd. *Mathematical Physiology: II: Systems Physiology*. Interdisciplinary Applied Mathematics. Springer, 2nd edition, 2008.

98. R. E. Klabunde. *Cardiovascular Physiology Concepts*. Wolters Kluwer Health, 2011.

99. D. Krause, K. Fackeldey, and R. Krause. A parallel multiscale simulation toolbox for coupling molecular dynamics and finite elements. 2012. to appear.

100. D. Krause, M. Potse, T. Dickopf, R. Krause, A. Auricchio, and F. W. Prinzen. Hybrid parallelization of a large-scale heart model. In R. Keller, D. Kramer, and J.-P. Weiss, editors, *Facing the Multicore-Challenge II*, volume 7174 of *Lecture Notes in Computer Science*, pages 120–132. Springer, 2012.

101. J. Lang. *Adaptive Multilevel Solution of Nonlinear Parabolic PDE Systems: Theory, Algorithm, and Applications*, volume 16 of *Lecture Notes in Computational Science and Engineering*. Springer, 2000.

102. B. Lee, S. F. McCormick, B. Philip, and D. J. Quinlan. Asynchronous fast adaptive composite-grid methods: Numerical results. *SIAM J. Sci. Comput.*, 25(2):682–700, 2003.

103. B. Lee, S. F. McCormick, B. Philip, and D. J. Quinlan. Asynchronous fast adaptive composite-grid method for elliptic problems: Theoretical foundations. *SIAM J. Numer. Anal.*, 42(1): 130–152, 2005.

104. G. T. Lines, P. Grottum, and A. Tveito. Modeling the electrical activity of the heart: A bidomain model of the ventricles embedded in a torso. *Comput. Vis. Sci.*, 5(4):195–213, 2003.

105. R. D. Loft, S. J. Thomas, and J. M. Dennis. Terascale spectral element dynamical core for atmospheric general circulation models. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 18–18, 2001.

106. C.-H. Luo and Y. Rudy. A model of the ventricular cardiac action potential. depolarization, repolarization, and their interaction. *Circ. Res.*, 68(6):1501–1526, 1991.

107. C.-H. Luo and Y. Rudy. A dynamic model of the cardiac ventricular action potential. I. Simulations of ionic currents and concentration changes. *Circ. Res.*, 74(6):1071–96, 1994.

108. C.-H. Luo and Y. Rudy. A dynamic model of the cardiac ventricular action potential. II. Afterdepolarizations, triggered activity, and potentiation. *Circ. Res.*, 74(6):1097–113, 1994.

109. X. Ma and N. Zabaras. An adaptive hierarchical sparse grid collocation algorithm for the solution of stochastic differential equations. *J. Comput. Phys.*, 228(8):3084–3113, 2009.

110. Y. Maday, C. Mavriplis, and A. T. Patera. Nonconforming mortar element methods – Application to spectral discretizations. In *Domain Decomposition Methods*, pages 392–418, 1989.

111. G. Mahinthakumar and F. Saied. A hybrid MPI-OpenMP implementation of an implicit finite-element code on parallel architectures. *Int. J. High Perform. C*, 16(4):371–393, 2002.

112. S. McCormick and D. Quinlan. Asynchronous multilevel adaptive methods for solving partial differential equations on multiprocessors: Performance results. *Parallel Comput.*, 12(2): 145–156, 1989.

113. G. R. Mirams, C. J. Arthurs, M. O. Bernabeu, R. Bordas, J. Cooper, A. Corrias, Y. Davit, S.-. Dunn, A. G. Fletcher, D. G. Harvey, M. E. Marsh, J. M. Osborne, P. Pathmanathan, J. Pitt-Francis, J. Southern, N. Zemzemi, and D. J. Gavaghan. Chaste: An open source C++ library for computational physiology and biology. *PLoS Comput. Bio.*, 9(3):e1002970, 03 2013.

114. A. A. Mirin, D. F. Richards, J. N. Glosli, E. W. Draeger, B. Chan, J.-L. Fattebert, W. D. Krauss, T. Oppelstrup, J. J. Rice, J. A. Gunnels, V. Gurev, C. Kim, J. Magerlein, M. Reumann, and H.-F. Wen. Toward real-time modeling of human heart ventricles at cellular resolution: Simulation of drug-induced arrhythmias. In *Proceedings of the 2012 ACM/IEEE conference on Supercomputing*, pages 2:1–2:11, 2012.

115. L. Mitchell, M. J. Bishop, E. Hötzl, A. Neic, M. Liebmann, G. Haase, and G. Plank. Modeling cardiac electrophysiology at the organ level in the peta flops computing age. *AIP Conf. Proc.*, 1281(1):407–410, 2010.

116. M. Munteanu and L. F. Pavarino. Decoupled Schwarz algorithms for implicit discretizations of nonlinear monodomain and bidomain systems. *Math. Model Meth. App. Sci.*, 19 (7):1065–1097, 2009.

117. M. Neumüller and O. Steinbach. Refinement of flexible space-time finite element meshes and discontinuous Galerkin methods. *Comput. Vis. Sci.*, 14(5):189–205, 2011.

118. S. Niederer, L. Mitchell, N. Smith, and G. Plank. Simulating a human heart beat with near-real time performance. *Front. Physiol.*, 2:14, 2011.

119. PARATEC. PARAllel Total Energy Code. http://www.nersc.gov/projects/paratec.

120. P. Pathmanathan, M. O. Bernabeu, R. Bordas, J. Cooper, A. Garny, J. M. Pitt-Francis, J. P. Whiteley, and D. J. Gavaghan. A numerical guide to the solution of the bidomain equations of cardiac electrophysiology. *Prog. Biophys. Mol. Bio.*, 102(2–3):136–155, 2010.

121. L. F. Pavarino and S. Scacchi. Multilevel additive Schwarz preconditioners for the bidomain reaction-diffusion system. *SIAM J. Sci. Comput.*, 31(1):420–443, 2008.

122. M. Pennacchio. A non-conforming domain decomposition method for the cardiac potential problem. In *Comput. in Cardiol.*, pages 537–540, 2001.

123. M. Pennacchio. The mortar finite element method for the cardiac "bidomain" model of extracellular potential. *J. Sci. Comput.*, 20(2):191–210, 2004.

124. G. Plank, R. A. Burton, P. Hales, M. J. Bishop, T. Mansoori, M. O. Bernabeu, A. Garny, A. J. Prassl, C. Bollensdorff, F. Mason, F. Mahmood, B. Rodriguez, V. Grau, J. E. Schneider, D. Gavaghan, and P. Kohl. Generation of histo-anatomically representative models of the individual heart: Tools and application. *Philos. T. R. Soc. A*, 367(1896):2257–2292, 2009.

125. B. Pope, B. Fitch, M. Pitman, J. Rice, and M. Reumann. Performance of hybrid programming models for multiscale, cardiac simulations: Preparing for petascale computation. *IEEE Trans. Biomed. Eng.*, 58(10):2965–2969, 2011.

126. M. Potse. Mathematical modeling and simulation of ventricular activation sequences: Implications for cardiac resynchronization therapy. *J. Cardio. Transl. Res.*, 5(2):146–158, 2012.

127. M. Potse, B. Dubé, J. Richer, A. Vinet, and R. M. Gulrajani. A comparison of monodomain and bidomain reaction-diffusion models for action potential propagation in the human heart. *IEEE Trans. Biomed. Eng.*, 53(12):2425–2435, 2006.

128. M. Potse, B. Dubé, and A. Vinet. Cardiac anisotropy in boundary-element models for the electrocardiogram. *Med. Biol. Eng. Comput.*, 47(7):719–729, 2009.

129. L. Priebe and D. J. Beuckelmann. Simulation study of cellular electric properties in heart failure. *Circ. Res.*, 82(11):1206–1223, 1998.

130. R. Rabenseifner, G. Hager, and G. Jost. Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 427–436, 2009.

131. C. A. Rendleman, V. E. Beckner, M. Lijewski, W. Crutchfield, and J. B. Bell. Parallelization of structured, hierarchical adaptive mesh refinement algorithms. *Comput. Vis. Sci.*, 3(3):147–157, 2000.

132. F. A. Roberge, A. Vinet, and B. Victorri. Reconstruction of propagated electrical activity with a two-dimensional model of anisotropic heart muscle. *Circ. Res.*, 58(4):461–475, 1986.

133. D. Rossinelli, M. Bergdorf, B. Hejazialhosseini, and P. Koumoutsakos. Wavelet-based adaptive solvers on multi-core architectures for the simulation of complex systems. In H. Sips, D. Epema, and H.-X. Lin, editors, *Euro-Par 2009 Parallel Processing*, volume 5704 of *Lecture Notes in Computer Science*, pages 721–734. Springer, 2009.

134. D. Rossinelli, B. Hejazialhosseini, D. G. Spampinato, and P. Koumoutsakos. Multicore/multi-GPU accelerated simulations of multiphase compressible flows using wavelet adapted grids. *SIAM J. Sci. Comput.*, 33(2):512–540, 2011.

135. S. Rush and H. Larsen. A practical algorithm for solving dynamic membrane equations. *IEEE Trans. Biomed. Eng.*, 25(4):389–392, 1978.

136. Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

137. O. Sahni, M. Zhou, M. S. Shephard, and K. E. Jansen. Scalable implicit finite element solver for massively parallel processing with demonstration to 160k cores. In *Proceedings of the 2009 ACM/IEEE conference on Supercomputing*, pages 68:1–68:12, 2009.

138. Hasan I. Saleheen and Kwong T. Ng. New finite difference formulations for general inhomogeneous anisotropic bioelectric problems. *IEEE Trans. Biomed. Eng.*, 44(9):800–809, 1997.

139. R. S. Sampath, S. S. Adavani, H. Sundar, I. Lashuk, and G. Biros. Dendro: Parallel algorithms for multigrid and AMR methods on 2:1 balanced octrees. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 18:1–18:12, 2008.

140. S. V. Sathe, R. Benney, R. D. Charles, E. Doucette, J. Miletti, M. Senga, K. R. Stein, and T. E. Tezduyar. Fluid-structure interaction modeling of complex parachute designs with the space-time finite element techniques. *Comput. Fluids*, 36(1):127–135, 2007.

141. B. Satish, J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.1, Argonne National Laboratory, 2010.

142. P. Schwartz, M. Barad, P. Colella, and T. Ligocki. A Cartesian grid embedded boundary method for the heat equation and Poisson's equation in three dimensions. *J. Comput. Phys.*, 211(2):531–550, 2006.

143. J. Southern, G. Plank, E. J. Vigmond, and J. P. Whiteley. Solving the coupled system improves computational efficiency of the bidomain equations. *IEEE Trans. Biomed. Eng.*, 56 (10):2404–2412, 2009.

144. J. Southern, G. J. Gorman, M. D. Piggott, P. E. Farrell, M. O. Bernabeu, and J. Pitt-Francis. Simulating cardiac electrophysiology using anisotropic mesh adaptivity. *J. Comput. Phys.*, 1 (2):82–88, 2010.

145. J. Southern, G. J. Gorman, M. D. Piggott, and P. E. Farrell. Parallel anisotropic mesh adaptivity with dynamic load balancing for cardiac electrophysiology. *J. Comput. Phys.*, 3(1–2):8–16, 2012.

146. R. Speck, D. Ruprecht, R. Krause, M. Emmett, M. Minion, M. Winkel, and P. Gibbon. A massively space-time parallel N-body solver. In *Proceedings of the 2012 ACM/IEEE conference on Supercomputing*, pages 92:1–92:11, 2012.

147. D. Stefanica. A numerical study of FETI algorithms for mortar finite element methods. *SIAM J. Sci. Comput.*, 23(4):1135–1160, 2001.

148. S. Sun and M. F. Wheeler. Mesh adaptation strategies for discontinuous Galerkin methods applied to reactive transport problems. In H.-W. Chu, M. Savoie, and B. Sanchez, editors, *Proceedings of the 2004 International Conference on Computing, Communication and Control Technologies*, volume 1, pages 223–228, 2004.

149. H. Sundar, G. Biros, C. Burstedde, J. Rudi, O. Ghattas, and G. Stadler. Parallel geometric-algebraic multigrid on unstructured forests of octrees. In *Proceedings of the 2012 ACM/IEEE conference on Supercomputing*, pages 43:1–43:11, 2012.

150. K. H. ten Tusscher and A. V. Panfilov. Alternans and spiral breakup in a human ventricular tissue model. *Am. J. Physiol.*, 291(3):H1088–H1100, 2006.

151. K. H. ten Tusscher, D. Noble, P. J. Noble, and A. V. Panfilov. A model for human ventricular tissue. *Am. J. Physiol.*, 286(4):H1573–H1589, 2004.

152. T. E. Tezduyar and S. V. Sathe. Enhanced-discretization space-time technique (EDSTT). *Comput. Methods Appl. M.*, 193(15–16):1385–1401, 2004.

153. T. E. Tezduyar, S. Aliabadi, and M. Behr. Enhanced-discretization interface-capturing technique (EDICT) for computation of unsteady flows with interfaces. *Comput. Methods Appl. M.*, 155(3–4):235–248, 1998.

154. T.E. Tezduyar, M. Behr, and J. Liou. A new strategy for finite element computations involving moving boundaries and interfaces – The deforming-spatial-domain/space-time procedure: I. The concept and the preliminary numerical tests. *Comput. Meth. Appl. Mech. Eng.*, 94(3): 339–351, 1992.

155. J. A. Trangenstein and C. Kim. Operator splitting and adaptive mesh refinement for the Luo-Rudy I model. *J. Comput. Phys.*, 196(2):645–679, 2004.

156. N. A. Trayanova. Defibrillation of the heart: Insights into mechanisms from modelling studies. *Experimental Physiology*, 91(2):323–337, 2006.

157. T. Tu, D. R. O'Hallaron, and O. Ghattas. Scalable parallel octree meshing for terascale applications. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, pages 4:1–4:15, 2005.

158. L. Tung. *A bi-domain model for describing ischemic myocardial D-C potentials*. PhD thesis, MIT, Cambridge, MA, 1978.

159. B. Van Straalen, P. Colella, D. T. Graves, and N. Keen. Petascale block-structured AMR applications without distributed meta-data. In E. Jeannot, R. Namyst, and J. Roman, editors, *Euro-Par 2011 Parallel Processing*, volume 6853 of *Lecture Notes in Computer Science*, pages 377–386. Springer Berlin Heidelberg, 2011.

160. M. Vázquez, R. Arís, G. Houzeaux, R. Aubry, P. Villar, J. Garcia-Barnés, D. Gil, and F. Carreras. A massively parallel computational electrophysiology model of the heart. *Int. J. Numer. Meth. Biomed. Eng.*, 27(12):1911–1929, 2011.

161. E. J. Vigmond, F. Aguel, and N. A. Trayanova. Computational techniques for solving the bidomain equations in three dimensions. *IEEE Trans. Biomed. Eng.*, 49(11):1260–1269, 2002.

162. E. J. Vigmond, R. Weber dos Santos, A. J. Prassl, M. Deo, and G. Plank. Solvers for the cardiac bidomain equations. *Prog. Biophys. Mol. Bio.*, 96(1–3):3–18, 2008.

163. M. S. Warren and J. K. Salmon. A parallel hashed oct-tree N-body algorithm. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 12–21, 1993.

164. T. Weinzierl and T. Köppl. A geometric space-time multigrid algorithm for the heat equation. *Numerical Mathematics: Theory, Methods and Applications*, 5(1):110–130, 2012.

165. T. Weinzierl and M. Mehl. Peano – A traversal and storage scheme for octree-like adaptive Cartesian multiscale grids. *SIAM J. Sci. Comput.*, 33(5):2732–2760, 2011.

166. M. Weiser, B. Erdmann, and P. Deuflhard. On efficiency and accuracy in cardioelectric simulation. In H.-G. Bock, F. Hoog, A. Friedman, et al., editors, *Progress in Industrial Mathematics at ECMI 2008*, volume 15 of *Mathematics in Industry*, pages 371–376. Springer Berlin Heidelberg, 2010.

167. J. P. Whiteley. Physiology driven adaptivity for the numerical solution of the bidomain equations. *Ann. Biomed. Eng.*, 35(9):1510–1520, 2007.

168. Jonathan P. Whiteley. An efficient numerical technique for the solution of the monodomain and bidomain equations. *IEEE Trans. Biomed. Eng.*, 53(11):2139–2147, 2006.

169. A. M. Wissink, R. D. Hornung, S. R. Kohn, S. S. Smith, and N. Elliott. Large scale parallel structured AMR calculations using the SAMRAI framework. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 6–6, 2001.

170. B. I. Wohlmuth. A residual based estimator for mortar finite element discretizations. *Numer. Math.*, 84(1):143–171, 1999.

171. B. I. Wohlmuth. A mortar finite element method using dual spaces for the Lagrange multiplier. *SIAM J. Numer. Anal.*, 38(3):989–1012, 2000.

172. B. I. Wohlmuth. *Discretization Techniques and Iterative Solvers Based on Domain Decomposition*, volume 17. Springer, 2001.

173. B. I. Wohlmuth and R. Krause. Monotone multigrid methods on nonmatching grids for nonlinear multibody contact problems. *SIAM J. Sci. Comput.*, 25(1):324–347, 2003.

174. B. I. Wohlmuth and R. H. Krause. A multigrid method based on the unconstrained product space for mortar finite element discretizations. *SIAM J. Numer. Anal.*, 39(1):192–213, 2002.

175. W. Ying and C. S. Henriquez. Adaptive mesh refinement for modeling cardiac electrical dynamics. *Chaos*, 2011. submitted.

176. H. Yu. Solving parabolic problems with different time steps in different regions in space based on domain decomposition methods. *Appl. Numer. Math.*, 30(4):475–491, 1999.

177. H. Yu. A local space-time adaptive scheme in solving two-dimensional parabolic problems based on domain decomposition methods. *SIAM J. Sci. Comput.*, 23(1):304–322, 2001.

178. G. W. Zumbusch. A sparse grid PDE solver; discretization, adaptivity, software design and parallelization. In H. P. Langtangen, A. M. Bruaset, and E. Quak, editors, *Advances in Software Tools for Scientific Computing*, volume 10 of *Lecture Notes in Computational Science and Engineering*, pages 133–177. Springer, 2000.