View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by OTHES

universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

Numerical Solution of the Generalised Poisson Equation on Parallel Computers

Angestrebter akademischer Grad

Magister der Naturwissenschaften (Mag. rer. nat.)

Verfasser: Hannes Grimm-Strele
Matrikel-Nummer: 0404659
Studienrichtung: A 405 Mathematik
Betreuer: Univ.-Prof. Dr. Herbert Muthsam

Wien, im März 2010

Danksagung

Ich danke Herrn Professor Muthsam für das interessante Thema meiner Diplomarbeit und die hilfreichen Ratschläge, die er mir bei der Durchführung gab, sowie für meine Aufnahme in seiner Forschungsgruppe ACORE. Allen Mitgliedern von ACORE danke ich für die gute Zusammenarbeit während der Anfertigung meiner Arbeit.

Einen wichtigen Beitrag leistete auch das Rechencluster Xian der Fakultät für Mathematik der Universität Wien, wo ich einen Teil der Rechenläufe durchführen konnte.

Mein Dank gilt allen, die direkt oder indirekt an der Erstellung dieser Arbeit mitgewirkt haben. Insbesondere danke ich Marcus Page, David Wimmesberger und Arno Mayrhofer für die vielen hilfreichen Korrektur- und Verbesserungsvorschläge, ebenso Johannes Temme für die Formatvorlage.

Einen ganz besonderen Anteil an dieser Arbeit haben meine Eltern. Ohne meine zwei Besuche bei ihnen in den letzten Wochen des Schreibens hätte ich niemals so schnell vorankommen können, wie es bei ihnen möglich war. Darüber hinaus hat mein Vater den wesentlichen Teil des Korrekturlesens durchgeführt und mir mit vielen Hinweisen geholfen.

Diese Arbeit wurde finanziell durch den Fonds zur Förderung der wissenschaftlichen Forschung (FWF) unterstützt.

Abstract

In this work, a method is presented to numerically solve the Generalised Poisson Equation

$$-\nabla \cdot (\kappa(x)\nabla u(x)) + c(x)u(x) = f(x)$$

on parallel computers. This type of partial differential equation arises in many different (astro-)physical contexts, two of which will be discussed shortly.

In scientific computing, parallel programming plays a decisive role since most problems are too complex to be solved on one single processing entity (PE). Therefore, algorithms must be developed which are suitable for parallel execution.

The Generalised Poisson Equation imposes particular challenges to the parallel program because its solution is non-local, i.e. the solution in one point is influenced by the whole computational domain. With the Schur Complement Method, the global solution within the framework of Domain Decomposition can be obtained by first solving a problem for the interface nodes and then independent problems for the inner domain of each PE.

This work is organised as follows:

- In **Chapter 2**, the analytical and physical background of this work is presented.
- The numerical methods which were used are described in **Chapter 3**. First, the equation is discretised by the Finite Element Method. The resulting linear system is then inverted using the Schur Complement Method. The interface problem is solved iteratively in parallel, whereas the local problem on every PE is solved by a (preconditioned) CG algorithm.
- Numerical results concerning the scaling and the implementation, espacially the choice of the parameters, are given in **Chapter 4**.
- In **Chapter 5**, possible extensions of this work and how the method could be further improved, are discussed.

In **Appendix A**, an implementation of the Finite Element Method in one, two and three dimensions over a rectangular, equidistant grid is presented.

Zusammenfassung

In dieser Diplomarbeit wird eine Methode zur numerischen Lösung der Verallgemeinerten Poissongleichung

$$-\nabla \cdot (\kappa(x)\nabla u(x)) + c(x)u(x) = f(x)$$

auf Parallelrechnern vorgestellt. Diese Form einer partiellen Differentialgleichung tritt in vielen verschiedenen (astro-)physikalischen Zusammenhängen auf. Zwei Beispiele werden in aller Kürze dargestellt werden.

Die meisten Aufgabenstellungen des wissenschaftlichen Rechnens sind zu komplex und ressourcenaufwendig, um auf einem einzelnen Rechner ausgeführt werden zu können. Daher ist es von enormer Bedeutung, Algorithmen zu entwickeln, die zum Parallelrechnen geeignet sind.

Da die Lösung der Verallgemeinerten Poissongleichung nicht-lokal ist – das bedeutet, dass die Lösung in einem Punkt vom gesamten Gebiet beeinflusst wird –, steht die Entwicklung einer Lösungsmethodik vor besonderen Schwierigkeiten, wenn man die globale Kommuniation möglichst gering halten will. Im Zusammenhang mit Parallelisierung durch Gebietszerlegung bietet die Schur-Komplement-Methode eine Möglichkeit zur Bewältigung dieses Problems. Dabei wird zuerst die Lösung der Gleichung auf den Randknoten berechnet und schließlich mit den zuvor bestimmten Werten als Randbedingung voneinander unabhängige Probleme auf den lokalen Gebieten gelöst.

Diese Arbeit ist wie folgt aufgebaut:

- In **Kapitel 2** wird die analytische und physikalische Grundlage vorgestellt.
- Die verwendeten Methoden zur numerischen Lösung der Verallgemeinerten Poissongleichung werden in **Kapitel 3** beschrieben. Zuerst wird die Gleichung mit der Finiten Elemente-Methode diskretisiert. Anschließend wird das resultierende lineare System mit der Schur-Komplement-Methode invertiert. Dabei wird zunächst die Lösung auf den Randknoten mittels eines iterativen Algorithmus bestimmt. Die lokale Lösung wird mit Hilfe eines präkonditionierten CG Algorithmus bestimmt.

- Numerische Daten zur Skalierung und zur Implementierung der Methode, insbesondere zur geeigneten Wahl der Parameter, sind in **Kapitel 4** zusammen gestellt.
- Schließlich wird in **Kapitel 5** ein Ausblick gegeben, wie diese Arbeit erweitert und die Methode verbessert werden könnte.

Im **Appendix A** wird eine Implementierung der Finiten Elemente-Methode in einer, zwei und drei Dimensionen über einem rechteckigen, äquidistanten Gitter vorgestellt.

Contents

1 Introduction **1**

2 The Generalised Poisson Equation **3**

2.1 Analysis 3

2.1.1 Preliminaries 3

2.1.2 Existence and Uniqueness of the Solution 5

2.1.3 Regularity of the Solution 7

2.1.4 Boundary Conditions 8

2.2 Applications in Astrophysics 10

2.2.1 Magnetohydrodynamics 10

2.2.2 Pressure Update for the Euler Equations 11

3 Numerical Methods **13**

3.1 Parallel Computing 14

3.1.1 The Message Passing Interface 15

3.1.2 Domain Decomposition 16

3.2 The Finite Element Method 16

3.2.1 Methodology 16

3.2.2 Boundary Conditions 18

3.3 The Conjugate-Gradient Algorithm 19

3.4 The Preconditioned CG Algorithm 20

3.4.1 The Symmetric Gauss-Seidel Preconditioning 22

3.4.2 The Incomplete Cholesky Decomposition 22

3.5 The Schur Complement Method 25

3.5.1 Methodology 26

3.5.2 Parallelisation 28

4 Numerical Results **35**

4.1 Test Case 1 35

4.2 Test Case 2 38

4.3 Test Case 3 41
4.4 Test Case 4 42

5 Outlook 45

A An Implementation of the Finite Element Method 47
A.1 On a Straight line 47
A.2 On a Rectangle 51
A.3 On a Cuboid 56

Curriculum Vitae 63

Chapter 1

Introduction

Subject of this diploma thesis is the numerical treatment of the Generalised Poisson Equation

$$-\nabla \cdot (\kappa(x)\nabla u(x)) + c(x)u(x) = f(x)$$

on a bounded domain Ω and the implementation of an algorithm to solve the equation on parallel computers. The most commonly known differential equation of this kind is Poisson's Equation $-\Delta u = f$. The Generalised Poisson Equation arises in many forms in the (astro)physical context.

There are already well-known and well-proven methods to solve these types of partial differential equations numerically. Due to the ongoing progress in computing technology, which leads to an enormous amount of available computation kernels, algorithms are needed which allow the full usage of the new computation resources. The Schur Complement Method, which is the main subject of this work, is one of these. Thereby, the global problem is split into several local problems of smaller dimension and one problem for the interface nodes which must be solved first. Finally, the global solution is assembled from the interface and the local problems.

In this diploma thesis, I present all methods and algorithms I used to write a solver for the Generalised Poisson Equation in Fortran90 which will be integrated in the code ANTARES. ANTARES is an hydrodynamic code developed at the University of Vienna which is described in [14]. With it simulations of the solar granulation, A stars, cepheids and the stellar interior can be performed. Furthermore, a solver for the magnetohydrodynamic equations is under development. The program is fully parallelised with MPI and partially with OpenMP. In this context, a fast and efficient parallel solver for the Generalised Poisson Equation is needed. The integration of the solver into ANTARES imposes certain requirements concerning e.g. the numerical grid.

Chapter 2

The Generalised Poisson Equation

2.1 Analysis

In this section I will investigate some properties of the Generalised Poisson Equation. First of all, some basic definitions are formulated and results from the general theory of partial differential equations are recapitulated. Then, the Generalised Poisson Equation is introduced and some main properties are deduced. In the end, I collect some examples from (astro-)physics where these types of partial differential equations arise. In the following, $\Omega \subset \mathbb{R}^n$ is always open and bounded and Γ is the boundary of the closure of Ω.

2.1.1 Preliminaries

The description is restricted to the main points. Please see [4, p. 239 – 266] for more details and proofs of the cited theorems.

Definition 2.1.1. Let $u \in L^1_{loc}(\Omega)$. If there exist functions $v_i \in L^2(\Omega), i = 1, \dots, n$, such that

$$\int_\Omega u \frac{\partial \phi}{\partial x_i} \, dx = - \int_\Omega v_i \phi \, dx \qquad \forall \phi \in C_c^\infty(\Omega), \tag{2.1.1}$$

we call $\nabla u = [v_1, \dots, v_n]^T$ the **weak gradient** of u.

For differentiable u, the weak gradient is identical to the classical gradient.

Definition 2.1.2. The standard inner product of the space $H^1(\Omega) \subset L^2(\Omega)$ is defined by

$$\langle u, v \rangle_{H^1(\Omega)} := \int_\Omega uv dx + \int_\Omega \nabla u \cdot \nabla v dx, \tag{2.1.2}$$

where ∇ must be understood in the weak sense. A function $u \in L^2(\Omega)$ belongs to $H^1(\Omega)$, if

$$\|u\|_{H^1(\Omega)} := \sqrt{\langle u, u \rangle_{H^1(\Omega)}} < \infty. \tag{2.1.3}$$

The space $H^1(\Omega)$ is called the **Sobolev space of order 1**.

Remark 2.1.3.

- $H^1(\Omega)$ contains all real-valued functions which are square integrable and whose weak derivatives are square integrable.
- One can analogously define the spaces $H^k(\Omega)$ for $k \in \mathbb{N}$. A function is in $H^k(\Omega)$ if all weak derivatives of order at most k are in $L^2(\Omega)$.
- The spaces $H^k(\Omega)$ are Hilbert spaces.

The following definition is taken from [4, p. 626].

Definition 2.1.4. Assume $k \in \mathbb{N}$. We say Γ is C^k if for each point $\vec{x_0} \in \Gamma$ there exist $r > 0$ and a C^k function $\gamma : \mathbb{R}^{n-1} \to \mathbb{R}$ such that — upon relabeling and reorienting the coordinate axes if necessary — we have

$$\Omega \cap B(\vec{x_0}, r) = \left\{ \vec{x} \in B(\vec{x_0}, r) | x_n > \gamma(x_1, \dots, x_{n-1}) \right\}. \tag{2.1.4}$$

Theorem 2.1.5. *Assume Γ is C^1. Then there exists a bounded linear operator*

$$T : H^1(\Omega) \to L^2(\Gamma),$$

such that

$$Tu = u|_\Gamma \text{ if } u \in H^1(\Omega) \cup C(\bar{\Omega}). \tag{2.1.5}$$

We call Tu the **trace** *of u on Γ.*

Definition 2.1.6. The space $H_0^1(\Omega)$ is a subspace of $H^1(\Omega)$ with

$$u \in H_0^1(\Omega) \Leftrightarrow u \in H^1(\Omega) \text{ and } u|_\Gamma = 0. \tag{2.1.6}$$

Theorem 2.1.7 (Poincaré)**.** *If $u \in H_0^1(\Omega)$, there exists a constant C such that*

$$\|u\|_{L^2(\Omega)} \leq C \|\nabla u\|_{L^2(\Omega)}. \tag{2.1.7}$$

Finally, we need the Riesz Representation Theorem which can be found in [4, p. 639].

Theorem 2.1.8 (Riesz). *Let H be a Hilbert space with inner product $\langle \cdot, \cdot \rangle_H$. For every bounded linear functional l on H, there exists a unique element $u \in H$, such that*

$$l(v) = \langle u, v \rangle_H \ \forall v \in H. \tag{2.1.8}$$

2.1.2 Existence and Uniqueness of the Solution

Now, the Generalised Poisson Equation is presented and existence and uniqueness of solutions (in the weak sense) are proven. Please look at [4] and [7] for a more general and rigorous treatment.

Let the differential operator L be given by

$$L[u] := -\nabla \cdot (\kappa \nabla u) + cu, \tag{2.1.9}$$

where κ and c are real-valued bounded functions in $\bar{\Omega}$ and $c(x) \geq 0$.

We consider the differential equation with the homogeneous Dirchlet boundary condition

$$L[u] = f \text{ on } \Omega \tag{2.1.10}$$

$$u = 0 \text{ on } \Gamma, \tag{2.1.11}$$

where $f \in L^2(\Omega)$. For $\kappa(x) = 1$, $c(x) = 0$ this is Poisson's Equation $-\Delta u = f$. In the general case, it is called the **Generalised Poisson Equation**. See 2.2 for some examples where these equations arise in an astrophysical context.

L is called **elliptic**, if

$$\exists \kappa_0 \in \mathbb{R} \text{ such that } \kappa(x) \geq \kappa_0 > 0 \ \forall x \in \bar{\Omega}. \tag{2.1.12}$$

In the following, L is always assumed elliptic. Furthermore, since $\kappa, c \in L^\infty(\Omega)$, there exist $\kappa_\infty, c_\infty > 0$ such that $\kappa(x) \leq \kappa_\infty$ and $c(x) \leq c_\infty$ for all $x \in \bar{\Omega}$.

u is called a **classical solution** of (2.1.10) and (2.1.11), if

$$u \in C^2(\bar{\Omega}), L[u] = f \text{ and } u|_\Gamma = 0. \tag{2.1.13}$$

If u is a classical solution and v is in $C^1(\bar{\Omega})$, we get by multiplying with v and integrating over Ω

$$\begin{aligned}\int_\Omega fv\ dx &= -\int_\Omega v\nabla\cdot(\kappa\nabla u)\ dx + \int_\Omega cuv\ dx\\ &= \int_\Omega \kappa\nabla u\cdot\nabla v\ dx + \int_\Omega cuv\ dx - \int_\Gamma \kappa v\frac{\partial u}{\partial\nu}\ ds,\end{aligned}$$

where ν is the unit outward normal. If $v|_\Gamma = 0$, this simplifies to

$$\int_\Omega fv\ dx = \int_\Omega \kappa\nabla u\cdot\nabla v\ dx + \int_\Omega cuv\ dx. \tag{2.1.14}$$

(2.1.14) is called the **weak formulation** of the boundary problem (2.1.10). But (2.1.14) is also valid if u and v are not in $C^2(\Omega)$, but in $H_0^1(\Omega)$. This leads directly to the following

Definition 2.1.9. $u \in H_0^1(\Omega)$ is called a **weak solution** of (2.1.10) and (2.1.11) if (2.1.14) is true for every $v \in H_0^1(\Omega)$.

Definition 2.1.10. The bilinear mapping $a(\cdot,\cdot)$ associated with the differential operator L is defined by

$$a : H_0^1(\Omega)\times H_0^1(\Omega)\to\mathbb{R}, a(u,v) = \int_\Omega \kappa\nabla u\cdot\nabla v\ dx + \int_\Omega cuv\ dx. \tag{2.1.15}$$

Furthermore, we define for given $f \in L^2(\Omega)$

$$l(v) := \langle f, v\rangle_{L^2(\Omega)} = \int_\Omega fv\ dx,\ \ v\in H_0^1(\Omega). \tag{2.1.16}$$

l is bounded since $\|l\| = \sup\left\{|l(v)| : \|v\|_{H_0^1(\Omega)} \le 1\right\} \le \|f\|_{L^2(\Omega)} < \infty$.

Lemma 2.1.11. *Let $u, v \in H_0^1(\Omega)$. There exist constants $\alpha, \beta > 0$ such that*

$$|a(u,v)| \le \alpha\,\|u\|_{H^1(\Omega)}\,\|v\|_{H^1(\Omega)}\ \text{and} \tag{2.1.17}$$

$$\beta\,\|u\|_{H^1(\Omega)}^2 \le a(u,u). \tag{2.1.18}$$

Proof.

$$\begin{aligned}|a(u,v)| &\le \int_\Omega \kappa\,|\nabla u|\,|\nabla v|\ \ dx + \int_\Omega c\,|uv|\ \ dx\\ &\le \max\{\kappa_\infty, c_\infty\}\,\|u\|_{H^1(\Omega)}\,\|v\|_{H^1(\Omega)}\\ &= \alpha\,\|u\|_{H^1(\Omega)}\,\|v\|_{H^1(\Omega)}\,,\end{aligned}$$

where $\alpha := \max\{\kappa_\infty, c_\infty\}$. This means that a is continuous on $H_0^1(\Omega)$.

Furthermore,

$$\begin{aligned} a(u,u) &= \int_\Omega \kappa \left|\nabla u\right|^2 \ dx + \int_\Omega c \left|u\right|^2 \ dx \\ &\geq \kappa_0 \int_\Omega \left|\nabla u\right|^2 \ dx = \kappa_0 \left\|\nabla u\right\|^2_{L^2(\Omega)} \\ &\overset{(2.1.7)}{\geq} \kappa_0 \frac{1}{C} \left\|u\right\|^2_{L^2(\Omega)}, \end{aligned}$$

since $c(x) \geq 0$. By defining $\beta := \frac{\kappa_0}{C}$ we get $\beta \left\|u\right\|^2_{L^2(\Omega)} \leq a(u,u)$. $\square$

Theorem 2.1.12. *Let* $\kappa, c \in L^\infty(\Omega)$*,* $0 \leq c(x) \leq c_\infty$ *and* $0 < \kappa_0 \leq \kappa(x) \leq \kappa_\infty$*. Then* (2.1.10) *together with* (2.1.11) *has for every* $f \in L^2(\Omega)$ *a unique weak solution* $u \in H^1_0(\Omega)$*.*

This theorem is a special case of the Lax-Milgram Theorem described in [4].

Proof. We can define a new inner product on $H^1_0(\Omega)$ by

$$\langle u, v\rangle_a := a(u,v), \ \ u,v \in H^1_0(\Omega), \tag{2.1.19}$$

to which we can apply the Riesz Representation Theorem. This is a inner product because of the previous lemma. Therefore, u from (2.1.8) is the desired unique solution.

$\square$

2.1.3 Regularity of the Solution

The theorem which was proven above garantuees only existence and uniqueness of an element of $H^1_0(\Omega)$ which fulfils (2.1.14), under the assumption that κ and c are in $L^\infty(\Omega)$. But this solution does not automatically solve the original problem (2.1.10), since there are second derivatives involved which do not exist in general for functions in $H^1_0(\Omega)$.
Given below are two theorems from [4, p. 317] where it is shown how the solution gets "smoother" the smoother the coefficient functions κ, c and the right hand side of (2.1.10) become.

Theorem 2.1.13. *Assume* $\kappa \in C^1(\bar{\Omega})$*,* $c \in L^\infty(\Omega)$ *and* $f \in L^2(\Omega)$*. Additionally,* $\Gamma \in C^2$*. Let* u *be the unique weak solution of*

$$\begin{aligned} L[u] &= f \ \ \textit{in } \Omega \\ u &= 0 \ \ \textit{on } \Gamma, \end{aligned}$$

then $u \in H^2(\Omega)$*.*

If κ is at least differentiable, u fulfils (2.1.10) in the weak sense. Furthermore one can prove

Theorem 2.1.14. *Let m be a nonnegative integer. Assume $\kappa, c \in C^{m+1}(\bar{\Omega})$ and $f \in H^m(\Omega)$. Additionally, $\Gamma \in C^{m+2}$. Let u be the unique weak solution of*

$$\begin{aligned} L[u] &= f \ \textit{in}\ \Omega \\ u &= 0 \ \textit{on}\ \Gamma, \end{aligned}$$

then $u \in H^{m+2}(\Omega)$.

In the case of Poisson's Equation, one can informally say that u "has two more derivatives than f has" if the boundary of Ω is smooth enough.

2.1.4 Boundary Conditions

Until now, we only considered homogeneous Dirichlet boundary conditions, i.e. we were looking for solutions u with $u|_\Gamma = 0$. Hereafter, the changes are shown that occur when moving to inhomogeneous Dirichlet, Neumann and periodic boundary conditions. These conditions can be mixed. Further details can be found in [7, p. 676].

Inhomogeneous Dirichlet boundary

Consider (2.1.10) with the **inhomogeneous Dirichlet** boundary condition

$$u|_\Gamma = g \text{ with } g \in L^2(\Gamma). \tag{2.1.20}$$

As before, we call $u \in H^1(\Omega)$ a weak solution of (2.1.14) with (2.1.20), if $u|_\Gamma = g$ and (2.1.14) is true for all $v \in H^1_0(\Omega)$.
Suppose we can find a function u_0 with $u_0|_\Gamma = g$, then $w = u - u_0$ solves the homogeneous problem

$$\begin{aligned} L[w] &= L[u] - L[u_0] = f - L[u_0] \text{ in } \Omega \\ w &= 0 \text{ on } \Gamma, \end{aligned}$$

due to the linearity of L. Now we can reconstruct the solution $u \in H^1(\Omega)$ of the inhomogeneous problem from w and u_0.
The inhomogeneous problem does not have a solution for every given boundary $g \in L^2(\Gamma)$.

Neumann boundary

Consider (2.1.10) with the **Neumann** boundary condition

$$\frac{\partial u}{\partial \nu}|_\Gamma = g \text{ with } g \in L^2(\Gamma). \tag{2.1.21}$$

Multiplying (2.1.10) with $v \in H^1(\Omega)$ and integration by parts yields

$$\begin{aligned}\int_\Omega fv\ dx &= -\int_\Omega v\nabla\cdot(\kappa\nabla u)\ dx + \int_\Omega cuv\ dx\\ &= \int_\Omega \kappa\nabla u\cdot\nabla v\ dx + \int_\Omega cuv\ dx - \int_\Gamma v\kappa\frac{\partial u}{\partial \nu}\ ds\\ &= \int_\Omega \kappa\nabla u\cdot\nabla v\ dx + \int_\Omega cuv\ dx - \int_\Gamma v\kappa g\ ds.\end{aligned}$$

The last term does not disappear any more, since we chose $v \in H^1(\Omega)$. Therefore, the weak formulation of (2.1.10) with Neumann boundary conditions is

$$\int_\Omega \kappa\nabla u\cdot\nabla v\ dx + \int_\Omega cuv\ dx = \int_\Omega fv\ dx + \int_\Gamma v\kappa g\ ds, \tag{2.1.22}$$

where u and v are in $H^1(\Omega)$. u is the weak solution of (2.1.10) with (2.1.21). Existence and uniqueness of this problem depend on c. Let $\kappa \in L^\infty(\Omega)$, $0 < \kappa_0 \le \kappa(x)$, $c \in L^\infty(\Omega)$, $f \in L^2(\Omega)$, $g \in L^2(\Gamma)$.

Lemma 2.1.15. *Assume* $c(x) \ge c_0 > 0$. *Then* (2.1.22) *has a unique solution* $u \in H^1(\Omega)$.

Lemma 2.1.16. *Assume* $c = 0$. *If*

$$\int_\Omega f(x)\ dx = -\int_\Gamma \kappa(s)g(s)\ ds, \tag{2.1.23}$$

there exist solutions of (2.1.10) *with* (2.1.21) *which differ by an additive constant. By postulating* $\int_\Omega u(x)\ dx = 0$, *the solution becomes unique.*

The solvability condition in the second case can be derived easily by setting $v = 1$ and $c = 0$ in (2.1.22).

Periodic boundary

If periodic boundary conditions in a certain direction are used, the upper boundary of the computational domain in this direction is neighbouring the lower boundary. In many applications, it is quite common to use this type of boundary at least in some directions. Therewith one avoids the necessity to set values for the Dirichlet or Neumann boundary conditions — which sometimes cannot easily be done — since the boundaries of the computational domain are now connected to each other. Then, the region Ω is not a subset of $\mathbb{R}^n$ any more. E.g. in two spatial dimensions, Ω with periodic boundary conditions in the second direction can be thought of as a cylinder, a two-dimensional manifold, as depicted in Figure 2.1. Periodic boundaries can easily be mixed with other types of boundary conditions. From the mathematical point of view, there are only few changes which affect more the numerical implementation than the analytical part.

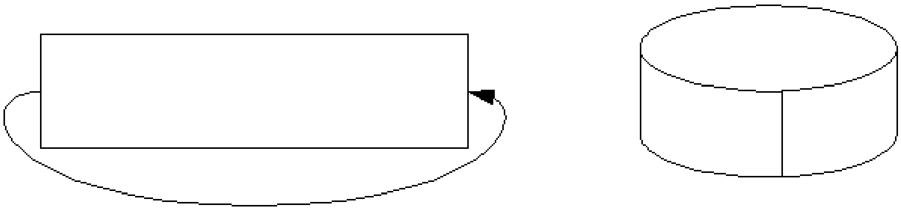

Figure 2.1: A two-dimensional rectangular domain with periodic boundary conditions can be thought of as the shell of a cylinder.

2.2 Applications in Astrophysics

The Generalised Poisson Equation (2.1.10) arises in many (astro)physical problems. In ANTARES, these are mainly the simulation of magnetohydrodynamics and conservative versions of the Euler equations. The derivation of (2.1.10) in these contexts is presented here in a nutshell.

2.2.1 Magnetohydrodynamics

If a fluid is (electrically) conductive, the Navier-Stokes equations do not sufficiently model the dynamics of the fluid, since it interacts with the electromagnetic field B. On the surface of the sun, all fluids are conductive, and therefore the **magnetohydrodynamic (MHD) equations** should be used to model the dynamics. The following description of the modelling procedure is mainly based on [11] and [9].

The MHD equations in three dimensions determine the time evolution of the fluid. Furthermore,

$$\nabla \cdot B = 0,$$

the electromagnetic field is divergence-free, since there are nor magnetic monopols nor electrical charges. From the analytic point of view, if the intial condition $B|_{t=0}$ is divergence-free, the time evolution does not violate this condition. But in the numerical simulation, this is not true any more. Instead, after some time steps,

$$\nabla \cdot B = err,$$

where *err* is an error function. One assumes that

$$err = \Delta\phi \text{ for some real-valued function } \phi,$$

which is of the form (2.1.10) with $\kappa(x) = 1$, $c(x) = 0$ and $f(x) = -err(x)$. By solving the above equation, we get a correction term ϕ for B such that

$$B_{\text{new}} := B - \nabla\phi$$

is divergence-free.

2.2.2 Pressure Update for the Euler Equations

The **Euler equations** govern the dynamics of a fluid without friction. Therein, the *momentum equation* in two dimensions is given by

$$\begin{pmatrix} \rho u \\ \rho v \end{pmatrix}_t + \underbrace{\begin{pmatrix} \rho u^2 \\ \rho uv \end{pmatrix}_x + \begin{pmatrix} \rho uv \\ \rho uv^2 \end{pmatrix}_y}_{\text{advection part}} + \underbrace{\begin{pmatrix} p_x \\ p_y \end{pmatrix}}_{\text{non-advection part}} = 0$$

with ρ being the density, $\vec{u} = (u, v)^T$ the velocity field and p the pressure. $\rho\vec{u}$ is called the *momentum* of the fluid. Using the advection part, an intermediate value

$$(\rho\vec{u})^\star = (\rho\vec{u})^n - \Delta t \left(\begin{pmatrix} \rho u^2 \\ \rho uv \end{pmatrix}_x + \begin{pmatrix} \rho uv \\ \rho uv^2 \end{pmatrix}_y \right)$$

of the updated momentum can be calculated. The momentum at time $t_{n+1} = t_n + \Delta t$ can now be determined with an Euler forward step by

$$\frac{(\rho\vec{u})^{n+1} - (\rho\vec{u})^\star}{\Delta t} = -\nabla p. \tag{2.2.1}$$

Since $\rho_t = -\nabla \cdot (\rho\vec{u})$, we can calculate $\rho^{n+1} = \rho^\star$. Dividing by ρ^{n+1} and taking the divergence yields

$$\nabla \cdot \vec{u}^{n+1} = \nabla \cdot \vec{u}^\star - \Delta t \nabla \cdot \left(\frac{\nabla p}{\rho^{n+1}} \right).$$

If the flow is incompressible, $\nabla \cdot \vec{u}^{n+1} = 0$, then

$$-\nabla \cdot \left(\frac{\nabla p}{\rho^{n+1}} \right) = -\frac{\nabla \cdot \vec{u}^\star}{\Delta t}$$

is of the form (2.1.10) with $\kappa = \frac{1}{\rho^{n+1}}$ and $f = -\frac{\nabla \cdot \vec{u}^\star}{\Delta t}$. Once p^{n+1} is determined in this way, $(\rho\vec{u})^{n+1}$ can be calculated by (2.2.1).

If the flow is compressible, more complicated calculations must be performed which can be found in [10], but again lead to an equation of the form (2.1.10).

Chapter 3

Numerical Methods

There are a lot of different methods to solve the Generalised Poisson Equation numerically but only few of them are suitable for the purpose of this work. The latter will be presented in this chapter.

To solve the Generalised Poisson Equation with given boundary conditions, (2.1.10) must be transformed into a linear system. Therefore, the equation must be discretised over a numerical grid. The most intuitive approach is to replace the first and second spatial derivative by the corresponding (central) difference quotient, i.e. in one spatial dimension with given grid $\{x_i : i = 1, \ldots, n\}$

$$\begin{aligned} \frac{\partial u}{\partial x} &\rightarrow \frac{u(x_{i+1}) - u(x_{i-1})}{x_{i+1} - x_{i-1}}, \\ \frac{\partial^2 u}{\partial x^2} &\rightarrow \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{(x_{i+1} - x_i)(x_i - x_{i-1})} \end{aligned}$$

and analogously for higher dimensions.

But for (2.1.10), the Finite Difference Method as it is described above leads to a nonsymmetric linear system since the first difference quotient is not symmetric. This has the big disadvantage that one cannot use the Conjugate Gradient algorithm to solve it. Therefore, I decided to use the Finite Element Method for the discretisation of the differential equation as described in 3.2 and Appendix A.

After the discretisation of the differential equation, the resulting sparse linear system must be solved. To solve sparse linear systems on a single processing entity (PE), there are a lot of algorithms. A description of the CG algorithm, which is one of the most common and effective methods, can be found in 3.3.

Matrices arising from the discretisation of elliptic operators usually have a high condition number and due to that, the CG algorithm converges slowly. Therefore, the idea of preconditioning was introduced to lower the condition number of the linear problem. This method will be presented in 3.4.

The dimension of the linear system equals the number of grid points in the simulation. In most applications, it is necessary to choose such a huge number of grid points that a single PE cannot handle the amount of data. Therefore the need of parallel computing arises, but this imposes special requirements on the numerical methods. Work must be split in several equal sized parts and transferred to each PE. From the local solutions, the global solution must be assembled. In section 3.1, the general design and idea of parallel computing will be introduced briefly.

The Schur Complement Method is a way to solve the linear system corresponding to (2.1.10) in parallel. It will be described in 3.5.

3.1 Parallel Computing

Moore's Law, according to which the number of transistors on a chip doubles every 18 months, probably will not be valid any more in some years. Instead, the number of cores on a chip will grow which enforces the need of parallel programming.

The practical relevance of (three-dimensional) hydrodynamic simulations depend on the resolution of the numerical grid and therefore on the number of grid points. E.g. for the investigation of turbulent flows on the solar surface, a resolution of 2 to 3 km is needed for a box of 6 Mm horizontal latitude. But increasing the number of grid points means simultaneously increasing the need of computation time and memory of the program. This leads to the necessity of parallel programming, since a single PE cannot execute the simulation in an acceptable amount of time nor can it provide enough memory for the simulation.

Parallel programming implies that the computational work is divided into several equal sized parts each of which are executed on separate PE's. From the resulting local solutions the global solution must be assembled. Therefore, parallelisation means distribution of work and data. The distributed work must be synchronised and communicated.

When designing a parallel program, there are two criteria which indicate a "good" parallelisation. The first one is load balancing, which means that the work is distributed such that the idle times of the PE's are minimised. The second one is optimal speedup, i.e. the program should run twice as fast if twice as many PE's are employed. Therefore, there should be as little effort for synchronisation and communication as possible. Good load balancing is a pre-requisite for optimal speedup.

Within the Fortran programming language, there are mainly two programming models to do this. In the **OpenMP model**, several PE's do work on data stored in a common memory. Therefore, no data decomposition is needed and no communication between the PE's takes place. Only the work is decomposed to the PE's.

In the **Message Passing Interface (MPI) model**, each PE has its own memory and therefore, work and data must be decomposed. MPI is much more powerful than OpenMP, but at the same time imposes much more work to the programmer, since the user must specify work and data distribution as well as the communication between the PE's. In 3.1.1, more information about the advantages and difficulties of MPI can be found.

Typically, to define the decomposition of data and work, domain decomposition is employed. According to this parallelisation technique, the computational domain is divided into as many subdomains as PE's are used. This will be described in 3.1.2.

Most of the information from this chapter is taken from [16].

3.1.1 The Message Passing Interface

Introduced in 1994, the **Message Passing Interface (MPI)** is a standard developed by the MPI Forum which provides a library of parallel routines for Fortran and C/C++ programs. In this library, operations for communication and message passing between the PE's are defined which can be used in the parallel program by subroutine calls (in Fortran90).

MPI is a distributed memory parallel programming model, i.e. each PE has its own memory and therefore its own data. For the designer of the parallel application, this means that he must not only specify the decomposition of computational work, but also the decomposition of data.

Therefore, MPI provides two modes of communication.

(i) **Point-to-Point Communication**. Only two PE's are involved, the first one as sender and the second one as receiver. The user must define the sending PE, the receiving PE, the message as well as the datatype and the length of the message.

(ii) **Global Communication**. In global operations, all PE's in the parallel simulation are involved. There are many types of global operations, e.g. one process sends a message to all others or one process collects information from all processes.

All MPI routines are completely described in the MPI Standard [5].

Since MPI is an open standard, there are a lot of implementations of MPI available. The most common ones are MPICH and OpenMPI. For the parallel program to be portable to different platforms, this must be considered in the design of the application.

The main advantage of MPI compared to other parallel architectures is that with MPI, the best speedup can be achieved, and hundreds or thousands of PE's can be used. On the other hand, the user must invest a considerable amount of work in the design of the parallel program.

3.1.2 Domain Decomposition

The Domain Decomposition Method is a way to define how work and data is decomposed for a MPI parallelisation. The latter can be most easily done by decomposing the global computational domain such that every PE has its own equal sized local domain.
On a rectangular or cuboidal domain, the number of PE's in the simulation is determined by the number of subdivisions in x, y and z direction. In most cases, the local domain of each PE is again a rectangle or a cuboid. For the solution of the Generalised Poisson Equation, all local solutions on the subdomains must interchange information. This can be done with the Schur Complement Method as described in 3.5.
Please see [15] for further information on this subject and for the implementation of this technique in ANTARES.

3.2 The Finite Element Method

With the **Finite Element Method (FEM)**, differential operators can be discretised and transformed to linear systems. Starting with the weak formulation of the problem, the key idea thereby is to choose a finite-dimensional ansatz space in which we look for an approximate solution.
The Finite Element Method has three main advantages compared to Finite Differences. First, more general geometries can be considered without much additional work. Second, the resulting linear system is always symmetric and positive definite. Third, there are only few restrictions concerning the smoothness of the functions involved. The resulting approximate solution will in general only fulfil the weak formulation of the problem. Please see [1] and [7] for details which will not be elaborated on the next pages.
An implementation of the Finite Element Method in one, two and three spatial dimensions is described in Appendix A.

3.2.1 Methodology

Consider (2.1.10) with homogeneous Dirichlet boundary conditions. Let Ω be a boundedregion, $\kappa, c \in L^\infty(\Omega)$ such that $0 < \kappa_0 \leq \kappa(x) \leq \kappa_\infty$ and $0 \leq c(x) \leq c_\infty$. Furthermore, $f \in L^2(\Omega)$. As already shown in 2.1.2, (2.1.10) can be reformulated to

$$\int_\Omega \kappa \nabla u \cdot \nabla v \; dx + \int_\Omega cuv \; dx = \int_\Omega fv \; dx,$$

where $u \in H_0^1(\Omega)$ is the weak solution of (2.1.10) if this holds true for all $v \in H_0^1(\Omega)$. As in the proof of 2.1.12, we define the bilinear form a by

$$a : H_0^1(\Omega) \times H_0^1(\Omega) \to \mathbb{R}, a(u,v) = \int_\Omega \kappa \nabla u \cdot \nabla v \; dx + \int_\Omega cuv \; dx, \tag{3.2.1}$$

which is symmetric, continuous on $H^1(\Omega)$ and there exists $\beta \in \mathbb{R}$, such that $\beta \left\| u \right\|_{H^1(\Omega)}^2 \leq a(u,u)$. Furthermore, l defined by

$$l : H_0^1(\Omega) \to \mathbb{R}, l(v) = \int_\Omega fv \; dx, \tag{3.2.2}$$

is a bounded linear functional on $H_0^1(\Omega)$. The weak solution of (2.1.10) can therefore be characterised as solution of

$$a(u,v) = l(v) \text{ for all } v \in H_0^1(\Omega). \tag{3.2.3}$$

Next, a finite-dimensional ansatz space $V_h \subset H_0^1(\Omega)$ is selected in which we look for an approximate solution. The most common example – and the space considered in this work – is the space of linear splines. Let $n \in \mathbb{N}$ be the dimension of V_h. The approximate solution $u_h \in V_h$ must fulfil

$$a(u_h, v_h) = l(v_h) \text{ for all } v_h \in V_h. \tag{3.2.4}$$

If $\{\phi_1, \ldots, \phi_n\}$ is a basis of V_h, then by setting $u_h = \sum_i u_i \phi_i$ and $v_h = \phi_j$ one gets directly the j-th line of the linear system $A\mathbf{u}_h = b$, where

$$A = \left[a(\phi_i, \phi_j)\right]_{ij} \in \mathbb{R}^{n \times n}, \;\; b = \left[l(\phi_j)\right]_j \in \mathbb{R}^n \tag{3.2.5}$$

and $\mathbf{u}_h = \left[u_1, \ldots, u_n\right]^T$ is the vector of the unknown coefficients of u_h in V_h. The matrix A is called **stiffness matrix**.

Lemma 3.2.1. *The matrix A defined by* (3.2.5) *is symmetric and positive definite.*

Proof. The symmetry of $A = [a_{ij}]$ is obvious:

$$a_{ij} = a(\phi_i, \phi_j) = a(\phi_j, \phi_i) = a_{ji}, \;\; i,j = 1, \ldots, n.$$

For $\mathbf{v} = \left[v_1, \ldots, v_n\right]^T \in \mathbb{R}^n$ and the associated function $v = \sum_i v_i \phi_i \in V_h$ it follows that

$$\langle \mathbf{v}, A\mathbf{v} \rangle_{\mathbb{R}^n} = \sum_{i,j=1}^n v_i v_j a(\phi_i, \phi_j) = a(\sum_{i=1}^n v_i \phi_i, \sum_{j=1}^n v_j \phi_j) = a(v,v).$$

Since $0 \leq \beta \left\| v \right\|_{H^1(\Omega)}^2 \leq a(v,v)$, A is positive definite. □

Therefore $A\mathbf{u}_h = b$ has a unique solution $\mathbf{u}_h \in \mathbb{R}^n$, from which we get directly the approximate solution $u_h \in V_h$.
The approximation error $\|u - u_h\|_{H^1(\Omega)}$ can be controlled by an appropriate choice of the ansatz space, which is shown in the following lemma.

Lemma 3.2.2 (Céa)**.** *Let u denote the weak solution of* (2.1.10) *with homogeneous Dirichlet boundary conditions and u_h its approximate solution in $V_h \subset H_0^1(\Omega)$. Then*

$$\|u - u_h\|_{H^1(\Omega)} \leq \frac{\alpha}{\beta} \inf_{v_h \in V_h} \|u - v_h\|_{H^1(\Omega)} , \tag{3.2.6}$$

with α, β from Lemma 2.1.11.

Proof. First we calculate:

$$\begin{aligned} a(u-u_h, u-u_h) &= a(u-u_h, u-v_h) + a(u, v_h - u_h) - a(u_h, v_h - u_h) \\ &= a(u-u_h, u-v_h) + l(v_h - u_h) - l(v_h - u_h) = a(u-u_h, u-v_h). \end{aligned}$$

Then

$$\begin{aligned} \beta \|u - u_h\|_{H^1(\Omega)}^2 &\leq a(u-u_h, u-u_h) = a(u-u_h, u-v_h) \\ &\leq \alpha \|u-u_h\|_{H^1(\Omega)} \|u - v_h\|_{H^1(\Omega)} , \end{aligned}$$

using the two estimates from lemma 2.1.11. After dividing by $\beta \|u - u_h\|_{H^1(\Omega)}$, the proposition is proven. ☐

3.2.2 Boundary Conditions

Until now, we only considered homogeneous Dirichlet boundary conditions. Now, we want to incorporate the considerations from 2.1.4 into the framework of the Finite Element Method.

Inhomogeneous Dirichlet boundary

As before, we choose a function $u_0 \in H^1(\Omega)$ such that $u_0|_\Gamma = g$. By setting $w = u - u_0 \in H_0^1(\Omega)$, we get an approximate solution $w_h \in V_h$ for the problem

$$a(w, v) = l(v) - a(u_0, v) \text{ for all } v \in H_0^1(\Omega). \tag{3.2.7}$$

From Lemma 3.2.2, for the approximation error it follows that

$$\|u - u_h\|_{H^1(\Omega)} = \|w - w_h\|_{H^1(\Omega)} \leq \frac{\alpha}{\beta} \inf_{v \in V_h} \|w - v\|_{H^1(\Omega)} . \tag{3.2.8}$$

Neumann boundary

If a Neumann boundary (2.1.21) is given, the right-hand side of (3.2.3) changes to

$$l(v) = \int_\Omega fv \; dx + \int_\Gamma g\kappa v \; ds. \tag{3.2.9}$$

Furthermore, the ansatz space V_h is now a subspace of $H^1(\Omega)$ and no longer of $H_0^1(\Omega)$. Again, one must distinguish between the cases $c(x) = 0$ and $c(x) \geq c_0 > 0$. In the first case the solution is not unique as described above.

Periodic boundary

If periodic boundary conditions are used, the ansatz space must be chosen accordingly. Let Ω be a two-dimensional rectangular manifold with homogeneous Dirichlet boundary conditions in the first and periodic boundary conditions in the second direction. Then Ω can be thought of as a cylinder, and

$$\begin{aligned} v \in H_0^1(\Omega) \Leftrightarrow & v \in L^2(\Omega), \nabla v \in L^2(\Omega), \\ & v(\cdot, y_1) = v(\cdot, y_2), \nabla v(\cdot, y_1) = \nabla v(\cdot, y_2), v(x_1, \cdot) = 0, v(x_2, \cdot) = 0, \end{aligned}$$

where $x_1, x_2, y_1, y_2 \in \mathbb{R}$ are the bounds of the domain in x respective y direction. For all basis functions in the ansatz space, it is now required that $\phi_i(\cdot, y_1) = \phi_i(\cdot, y_2)$.

3.3 The Conjugate-Gradient Algorithm

The **Conjugate-Gradient (CG) algorithm** is an iterative method to solve linear systems $Au = b$, where $A \in \mathbb{R}^{n \times n}$ is symmetric and positive definite and $u, b \in \mathbb{R}^n$. For sparse systems of large dimension, it is more suited than direct methods such as the LU or the Cholesky decomposition since it requires less memory and is in most cases much faster.

Instead of inverting the system directly, the quadratic functional

$$\Phi(u) = \frac{1}{2}\langle Au, u\rangle - \langle u, b\rangle, \tag{3.3.1}$$

which has an unique minimum since A is positive definite, is minimised. Given an initial guess $u^{(0)}$, a better approximation to the solution is found iteratively by the ansatz $u^{(k+1)} = u^{(k)} + \alpha_k d^{(k)}$, where the vectors $d^{(k)}$ are orthogonal with respect to the A inner product, i.e.

$$\langle d^{(k)}, d^{(l)} \rangle_A := \langle A d^{(k)}, d^{(l)} \rangle = 0 \text{ for } k \neq l, \tag{3.3.2}$$

and the step width α_k is given by $\alpha_k = \frac{\left\| r^{(k)} \right\|^2}{\langle d^{(k)}, d^{(k)} \rangle_A}$, where $r^{(k)} := b - Au^{(k)}$ is the residual. The algorithm 1 arrives at the unique exact solution u after at most n iterations. More information can be found in [17] or [7].

Algorithm 1 The CG algorithm.

```
Let $Au^{(0)} = b$ be given.
$r^{(0)} = b - Au^{(0)}$, $d^{(0)} = r^{(0)}$, $k = 0$
while $\left\| r^{(k)} \right\| > tol$ do
    $k = k + 1$
    $\alpha_k = \left\| r^{(k)} \right\|^2 / \langle Ad^{(k)}, d^{(k)} \rangle$
    $u^{(k+1)} = u^{(k)} + \alpha_k d^{(k)}$
    $r^{(k+1)} = r^{(k)} - \alpha_k A d^{(k)}$
    $\beta_k = \left\| r^{(k+1)} \right\|^2 / \left\| r^{(k)} \right\|^2$
    $d^{(k+1)} = r^{(k+1)} + \beta_k d^{(k)}$
end while
```

The norm of the residual $\left\| r^{(k)} \right\|$ is not an adequate way to measure the approximation error, since it can be changed arbitrarily by rescaling the equation as described in [3, p. 57]. Numerical experiments by the author indicate that a good choice for *tol* is $1.0 \cdot 10^{-16} \cdot \left\| r^{(0)} \right\|$ such that the relative error is reduced significantly, which is a quite common way to define the stop criterion. E.g. in [13], it is proposed to set *tol* to $1.0 \cdot 10^{-16} \cdot \left\| r^{(0)} \right\|$.
The convergence speed of algorithm 1 corresponds directly to the condition number of A as stated in [7, p. 309]. The runtime costs of the algorithm depend mainly on the computation time of the matrix-vector product $Ad^{(k)}$ and the inner product.

3.4 The Preconditioned CG Algorithm

Since the convergence speed of algorithm 1 depends on the condition number of A, one can try to accelerate the convergence by solving the system

$$M^{-1} A u = M^{-1} b, \tag{3.4.1}$$

if A and M are symmetric and positive definite. This system has the same solution as the original one, but, if M is chosen wisely,

$$cond(M^{-1}A) < cond(A). \tag{3.4.2}$$

$M^{-1}A$ is symmetric and positive definite with respect to the M inner product

$$\langle u, v \rangle_M := \langle Mu, v \rangle = \langle u, Mv \rangle. \tag{3.4.3}$$

If one replaces the usual Euclidean inner product in algorithm 1 by the M inner product, the algorithm 2 is obtained. In there, $\langle \cdot, \cdot \rangle$ still denotes the Euclidean inner product.

Algorithm 2 The preconditioned CG (PCG) algorithm.

```
Let $Au^{(0)} = b$ be given.
$r^{(0)} = b - Au^{(0)}$
Solve $Ms^{(0)} = r^{(0)}$
$d^{(0)} = s^{(0)}$, $k = 0$
while $\langle r^{(k)}, s^{(k)} \rangle > tol$ do
    $k = k + 1$
    $\alpha_k = \langle r^{(k)}, s^{(k)} \rangle / \langle d^{(k)}, Ad^{(k)} \rangle$
    $u^{(k+1)} = u^{(k)} + \alpha_k d^{(k)}$
    $r^{(k+1)} = r^{(k)} - \alpha_k A d^{(k)}$
    Solve $Ms^{(k+1)} = r^{(k+1)}$
    $\beta_k = \langle r^{(k+1)}, s^{(k+1)} \rangle / \langle r^{(k)}, s^{(k)} \rangle$
    $d^{(k+1)} = s^{(k+1)} + \beta_k d^{(k)}$
end while
```

In the preconditioned algorithm, the approximation error can be estimated by $\langle r^{(k)}, s^{(k)} \rangle$, as proposed in [3, p. 281].

Compared to algorithm 1, it is now additionally necessary to solve the system $Ms^{(k+1)} = r^{(k+1)}$ in every iteration. Therefore, it is essential that M is easily invertible. The gain in convergence speed by preconditioning, which results in less iterations needed to reach the stop criterion, must more than compensate the additional work in algorithm 2, i.e. the determination of M and its repeated inversion.

Therefore, the main problem of preconditioning is to find a matrix M which is symmetric, positive definite, easily invertible and in some sense similar to A. The best preconditioner would be A^{-1} because then we had convergence in one step, but in most cases we cannot calculate the inverse of A with reasonable effort. Setting $M = I$, where $I \in \mathbb{R}^{n \times n}$ is the identity matrix, we arrive at the CG algorithm.

In the following, two possible and well-tested choices for M are presented.

3.4.1 The Symmetric Gauss-Seidel Preconditioning

Let $A \in \mathbb{R}^{n \times n}$ be symmetric and positive definite. Then A can be written as

$$A = D - L^T - L, \tag{3.4.4}$$

where D is the diagonal matrix with the same diagonal entries as A and L is a lower triangle matrix with zeroes on the main diagonal and the negative entries of A elsewhere. For the **Symmetric Gauss-Seidel (SGS) Preconditioning**, one sets

$$M = (D - L)D^{-1}(D - L^T). \tag{3.4.5}$$

M can be inverted directly since it has Cholesky factor[1] $(D - L)D^{-\frac{1}{2}}$.
By

$$M = DD^{-1}D - LDD^{-1} - D^{-1}DL^T + LD^{-1}L^T = A + LD^{-1}L^T,$$

it is obvious that M is in a non-formal way similar to A and therefore also its inverse to A^{-1}.
The above M is a good preconditioner for A since no additional effort is imposed in determining M and it can be inverted easily because we know its Cholesky factor. Furthermore, imposing periodic boundary conditions is straightforward. It is not required that A is sparse or has a small bandwidth.

3.4.2 The Incomplete Cholesky Decomposition

We begin with a definition of the **Cholesky decomposition** of matrix A.

Definition 3.4.1. Let $A \in \mathbb{R}^{n \times n}$ be given. A factorisation $A = LL^T$, where L is an lower triangular matrix, is called **Cholesky decomposition** of A.

Lemma 3.4.2. *If $A \in \mathbb{R}^{n \times n}$ is symmetric and positive definite, there exists a Cholesky decomposition of A.*

See [7, p. 61] for a proof of this lemma.
The Cholesky decomposition of a symmetric and positive definite matrix A has the same bandwidth as A, but all entries within the bandwidth are non-zero in general. The matrix A determined according to 3.2 is sparse, but has a huge bandwidth. Thus it is not advisible to use the Cholesky decomposition to solve the linear system $Au = b$.

---

[1] $L \in \mathbb{R}^{n \times n}$ is called the **Cholesky factor** of $A \in \mathbb{R}^{n \times n}$ if L is a lower triangle matrix and $A = LL^T$. Since a triangle matrix can be inverted by one matrix-vector multiplication, once the Cholesky factor of A is known, A can be inverted directly in two steps.

Definition 3.4.3. Let $A \in \mathbb{R}^{n\times n}$ be given. The **sparsity pattern** P_A of A is defined by

$$P_A := \{(i,j) : a_{ij} \neq 0\}\,,\ i,j \in \{1,\dots,n\}\,. \tag{3.4.6}$$

Alternatively, the entry at position (i,j) of the Cholesky decomposition is calculated only if $(i,j) \in P_A$. This leads to the **Incomplete Cholesky Decomposition (ICD)** which has the same amount of non-zero entries and the same sparsity pattern as A and can be used as a preconditioner in algorithm 2. Its determination is described in algorithm 3. In fact, the (complete) Cholesky decomposition is in some sense a limit case of the incomplete decomposition, since by replacing P_A in algorithm 3 by $\{(i,j) : 1 \le i,j \le n\}$ we arrive at the complete decomposition. This observation motivates the following idea first proposed by [12].

Algorithm 3 Determination of the Incomplete Cholesky Decomposition (see [13, p. 212]).

```
Let $A = [a_{ij}]_{ij}$ be given and $L = [l_{ij}]_{ij}$ be a lower triangle matrix.
for $k = 1$ to $n$ do
    $l_{kk} = \left( a_{kk} - \sum_{\substack{j=1 \\ (k,j)\in P_A}}^{k-1} l_{kj}^2 \right)^{\frac{1}{2}}$
    for $i = k+1$ to $n$ do
        if $(k,i) \in P_A$ then
            $l_{ik} = \frac{1}{l_{kk}} \left( a_{ik} - \sum_{\substack{j=1 \\ (i,j)\in P_A,(k,j)\in P_A}}^{k-1} l_{ij} l_{kj} \right)$
        end if
    end for
end for
```

In the two-dimensional case with Dirichlet boundary conditions on all sides, the matrix A from the discretisation of the differential operator L over a rectangular grid is of a very regular structure, as shown in Figure 3.1. P_A only consists of five diagonals, so that there are only three diagonals where L from ICD has non-zero entries. Calculating the product LL^T we get a matrix K_1 which has non-zero entries on seven diagonals. Using its sparsity pattern P_{K_1} instead of P_A in algorithm 3, we get a matrix L_1 which has five diagonals with non-zero entries. We suppose that the product $(L_1 L_1^T)^{-1}$ is a better approximation to A^{-1} then $(LL^T)^{-1}$ because it is a better approximation to the complete Cholesky decomposition. This which will be verified by numerical experiments in REF!!!!. This procedure can be repeated, and thereby better preconditioners for A can be obtained. The main drawback is the need for additional memory and the time spent in the determination of the preconditioner. The number of additional diagonals in the sparsity pattern of the preconditioner L is usually referred to as number of fill-in η. Once η is fixed, $\lfloor \frac{\eta+1}{2} \rfloor$ diagonals are filled above the outer diagonal of A, whereas the remaining ones are filled below the inner diagonal. If the maximal number of fill-in is reached, the (complete) Cholesky decomposition is calculated, which is not effective except for very small systems.

Therefore, it is up to the user to decide how many additional diagonals should be included in the calculation of the preconditioner. Numerical experiments suggest for matrices with regular pattern as shown in Figure 3.1, that the bigger the distance to the outer and the inner diagonal of A gets, the smaller is the magnitude of the entries of the Cholesky decomposition.

This does not work in general if periodic boundary conditions are used because then A has entries away from the five diagonals depicted in Figure 3.1. Additional effort has to be made to cover this case, which is described in [12, p. 142 – 145]. There the three-dimensional case is discussed as well.

The effect of preconditioning as well as other relevant numerical data is depicted in chapter 4.

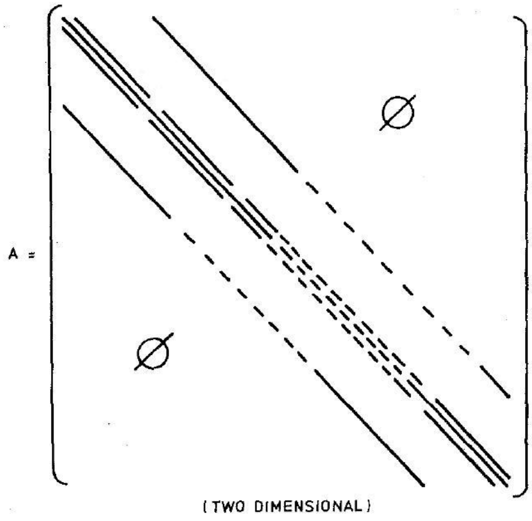

Figure 3.1: Sparsity pattern of A in two dimensions (from [12, p. 136]).

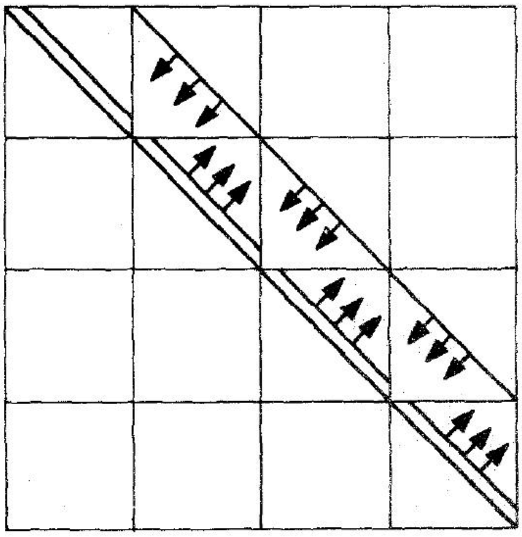

Figure 3.2: The magnitude of the entries of the Cholesky decomposition gets smaller along the arrows (from [12, p. 141]).

For matrices with a less regular sparsity pattern, this procedure will not generally lead to good preconditioners. Other criteria than "the position (i, j) is in P_A" may be more suitable to decide whether a distinct position should be included in the incomplete decomposition. For more details see [17, p. 296 – 320], [8] or [2]. Fortunately, the method described in [12] is well suited for the matrices occurring in this work, as it is stated e.g. in [8, p. 7].

To illustrate the above considerations, in Figure 3.3 the condition numbers of the matrix A corresponding to the discretisation of $L = -\Delta$ over a rectangular equidistant grid and of two preconditioned systems are plotted with *MATLAB*. The first preconditioner is the Incomplete Cholesky Decomposition as described in algorithm 3 and the second one ICD with some additional fill-in. Two main observations can be made:

- The condition number increases rapidly when the number of grid points is increased.
- The preconditioners lower the condition number essentially.

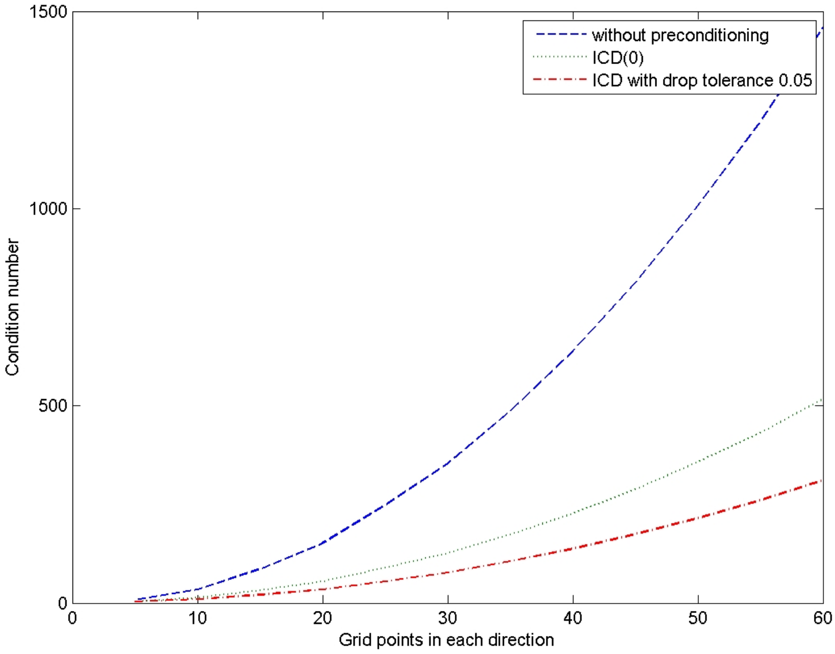

Figure 3.3: Condition number of the matrix corresponding to the discretisation of $L = -\Delta$ on an equidistant grid and of the preconditioned matrices with ICD(0) and drop tolerance 0.05 (see *MATLAB help* for details).

3.5 The Schur Complement Method

Using the Finite Element Method described in 3.2, the equation (2.1.10) is transformed into a linear system $Au = b$ the dimension of which is the number of grid points used in the numerical simulation. When this number gets large such that a single processing entity cannot provide enough memory or cannot solve the problem e.g. by the CG algorithm

described in 3.3 in an adequate amount of time, the work must be split on several PE's. Thereby, the main goals are to minimise the parallelisation overhead and to obtain a good scalability.
The **Schur Complement Method** is one way to achieve this. The procedure will be described in the next paragraphs. Further details can be obtained from [6, p. 200–228] and [17, p. 451–465].

3.5.1 Methodology

The main goal of this section is to develop a method to transform the linear problem

$$Au = b \text{ with } A \in \mathbb{R}^{n\times n}, u, b \in \mathbb{R}^n,\ n \in \mathbb{N}, \tag{3.5.1}$$

which should be solved for given A and right-hand side b, into (several) systems of smaller dimension.
We begin with a definition of the **Schur Complement Matrix**.

Definition 3.5.1. Let $A \in \mathbb{R}^{n\times n}$ be given, $n = n_1 + n_2$. If A is of the form

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \tag{3.5.2}$$

with certain submatrices $A_{1,1} \in \mathbb{R}^{n_1\times n_1}, A_{1,2} \in \mathbb{R}^{n_1\times n_2}, A_{2,1} \in \mathbb{R}^{n_2\times n_1}$ and $A_{2,2} \in \mathbb{R}^{n_2\times n_2}$ such that $A_{1,1}$ is invertible, one can define the **Schur Complement Matrix** $S_A \in \mathbb{R}^{n_2\times n_2}$ by

$$S_A = A_{2,2} - A_{2,1}A_{1,1}^{-1}A_{1,2}. \tag{3.5.3}$$

The system $Au = b$ from above is equivalent to

$$\begin{pmatrix} I & 0 \\ A_{2,1}A_{1,1}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{1,1} & A_{1,2} \\ 0 & S_A \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \tag{3.5.4}$$

where we set $u = (u_1, u_2)^T, b = (b_1, b_2)^T$ with $u_1, b_1 \in \mathbb{R}^{n_1}$, $u_2, b_2 \in \mathbb{R}^{n_2}$ and I is the identity matrix of suitable dimension.
Introducing the temporary variable $v = (v_1, v_2)^T$, we can reformulate $Au = b$ to

$$\begin{aligned} v_1 &= b_1, & v_2 &= b_2 - A_{2,1}A_{1,1}^{-1}v_1, \\ u_2 &= S_A^{-1}v_2 \ (\textit{Schur Complement Equation}), & u_1 &= A_{1,1}^{-1}(v_1 - A_{1,2}u_2). \end{aligned}$$

If S_A is invertible, these systems can be solved successively. The dimension of the systems is n_1 or n_2, whereby we succeeded in reducing the dimension of the linear systems to be solved.
We already remarked that the stiffness matrix A from 3.2 defined by (3.2.5) is always symmetric and positive definite. We state that if a matrix is symmetric and positive definite, it is invertible. Furthermore, A^{-1} as well as $A_{1,1}$ are symmetric and positive definite under this assumption.

Lemma 3.5.2. *Suppose A is symmetric and positive definite and define S_A by* (3.5.3)*. Then the following is true:*

(i) S_A is also symmetric and positive definite.

(ii) If $w \in \mathbb{R}^{n_2}$, then $S_A^{-1}w = R_2 A^{-1}(0,w)^T$, where $R_2(w_1,w_2)^T = w_2$.

Proof. From

$$det(A) = det\begin{pmatrix} I & 0 \\ A_{2,1}A_{1,1}^{-1} & I \end{pmatrix} \cdot det\begin{pmatrix} A_{1,1} & A_{1,2} \\ 0 & S_A \end{pmatrix} = 1 \cdot det(A_{1,1}) \cdot det(S_A)$$

it follows directly that S_A is invertible since A and $A_{1,1}$ are. Furthermore,

$$\begin{aligned} A^{-1} &= \begin{pmatrix} A_{1,1}^{-1} & -A_{1,1}^{-1}A_{1,2}S_A^{-1} \\ 0 & S_A^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -A_{2,1}A_{1,1}^{-1} & I \end{pmatrix} \\ &= \begin{pmatrix} A_{1,1}^{-1} + A_{1,1}^{-1}A_{1,2}S_A^{-1}A_{2,1}A_{1,1}^{-1} & -A_{1,1}^{-1}A_{1,2}S_A^{-1} \\ -S_A^{-1}A_{2,1}A_{1,1}^{-1} & S_A^{-1} \end{pmatrix}. \end{aligned}$$

Now we can proof the two claims separately.

(i) Since A^{-1} is symmetric and positive definite, from the above calculation it follows that S_A is so again.

(ii) We calculate

$$R_2 A^{-1}\begin{pmatrix} 0 \\ w \end{pmatrix} = R_2 \begin{pmatrix} -A_{1,1}^{-1}A_{1,2}S_A^{-1}w \\ S_A^{-1}w \end{pmatrix} = S_A^{-1}w.$$

□

Therefore, one can use the CG algorithm described in 3.3 to solve the systems mentioned above. We summarise these considerations in algorithm 4.

Algorithm 4 The serial Schur Complement Method.

```
Let $A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}$ be symmetric and positive definite.
$u = (u_1, u_2)^T$, $b = (b_1, b_2)^T$
$v_1 = b_1$
$v_2 = b_2 - A_{2,1} A_{1,1}^{-1} v_1$
$u_2 = S_A^{-1} v_2$
$u_1 = A_{1,1}^{-1} (v_1 - A_{1,2} u_2)$
```

Since we explicitly know the matrix $A_{1,1}$ or can at least determine it without much effort, there is no problem in solving it. This can be achieved either by the CG algorithm or by calculating its Cholesky decomposition, if the system is small enough such that this can done with reasonable computation costs. In contrast, we initially do not have any information about S_A. There are two possibilities to solve the Schur Complement Equation:

(i) S_A can be determined *explicitly* by (3.5.3). This is quite expensive since the linear system $A_{1,1}u_1 = b_1$ most be solved n_2 times. S_A is not sparse in general. After the initialisation of S_A, the CG algorithm or a Cholesky decomposition could be used to solve the system.

(ii) If the CG algorithm is employed to solve $S_A u_2 = v_2$, every step is a matrix-vector multiplication with S_A. Therefore, only the result of the application of S_A to a vector is needed.

In the following, we will mainly consider the second possibility since the explicit determination of S_A is too cumbersome in most cases. Especially in the context of parallelisation it will be obvious that the (preconditioned) CG algorithm is the best way to solve the Schur Complement Equation.

3.5.2 Parallelisation

In the previous paragraph, we split the original problem into two systems with smaller dimensions which must be solved subsequently. The next target is to make algorithm 4 suitable for parallel computing.

As already stated, in algorithm 4 linear systems with $A_{1,1}$ and S_A must be solved. This can be done in parallel, if $A_{1,1} \in \mathbb{R}^{n_1}$ has block structure, i.e.

$$A_{1,1} = \begin{pmatrix} D_1 & & & 0 \\ & D_2 & & \\ & & \ddots & \\ 0 & & & D_p \end{pmatrix}, \tag{3.5.5}$$

where $p \in \mathbb{N}$, $D_1 \in \mathbb{R}^{n_{11} \times n_{11}}, \dots, D_p \in \mathbb{R}^{n_{1p} \times n_{1p}}$ and $n_{11} + \cdots + n_{1p} = n_1$. Analogously we write

$$\begin{aligned} A_{1,2} &= (B_1, B_2, \dots, B_p)^T \text{ and} \\ A_{2,1} &= (C_1, C_2, \dots, C_p), \end{aligned}$$

where $B_1 \in \mathbb{R}^{n_{11} \times n_2}, \dots, B_p \in \mathbb{R}^{n_{1p} \times n_2}$ and $C_1 \in \mathbb{R}^{n_2 \times n_{11}}, \dots, C_p \in \mathbb{R}^{n_2 \times n_{1p}}$. Consequently,

$$S_A = A_{2,2} - \sum_{j=1}^{p} C_j D_j^{-1} B_j, \tag{3.5.6}$$

and for $w = (w_1, \dots, w_p)^T \in \mathbb{R}^{n_1}$ it follows that

$$A_{1,1} w = \begin{pmatrix} D_1 & & & 0 \\ & D_2 & & \\ & & \ddots & \\ 0 & & & D_p \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \end{pmatrix} = \begin{pmatrix} D_1 w_1 \\ D_2 w_2 \\ \vdots \\ D_p w_p \end{pmatrix}. \tag{3.5.7}$$

If the CG or the PCG algorithm is employed to solve the linear systems, most time is spent in the matrix-vector multiplication with $A_{1,1}$ and S_A. With the above considerations, both operations can be done in parallel employing p PE's. Furthermore, the multiplication with $A_{1,2}$ and $A_{2,1}$ can be executed independently by each PE. This is summarised in algorithm 5 where the FOR loops can be executed independently.

Algorithm 5 The parallel Schur Complement Method.

```
for $j = 1$ to $p$ do
    Solve $D_j z_{1j} = b_{1j}$
    Determine $C_j z_{1j}$
end for
$v_2 = b_2 - \sum_{j=1}^{p} C_j z_{1j}$
Solve the Schur Complement Equation $S_A u_2 = v_2$ in parallel
for $j = 1$ to $p$ do
    Solve $D_j u_{1j} = b_{1j} - B_j u_2$
end for
```

Obviously, the global summation $\sum_{j=1}^{p} C_j z_{1j}$ and the sending of the values of u on the interface nodes are the only communication operations in this algorithm, except for the (yet undescribed) part where the Schur Complement Equation is solved.

It now remains to transform A from Section 3.2 into the desired structure. The parallel algorithm to solve the Schur Complement Equation will be a consequence of the following considerations.

When the Domain Decomposition Method from paragraph 3.1.2 is employed, we have to transform A in the form depicted above. The key idea thereby is a suitable renumbering of the nodes. The procedure in two spatial dimensions can be outlined by the following steps:

(i) Identification of the physical boundaries. If Dirichlet boundary conditions are given, the corresponding nodes may not be considered in the following.

(ii) Identification of the domain boundaries. The nodes on one side of the boundary are selected as **interface nodes**, e.g. the last nodes of the lower domain.

(iii) Renumbering of all nodes starting with the **inner nodes** (i.e. all nodes except of the interface nodes) of one domain, then switching to the next domain and so on. In the end, all interface nodes are numbered subsequently.

This numbering applied to a grid with 10 × 8 nodes and four PE's is illustrated in Figure 3.4. The resulting matrix A is depicted in Figure 3.5. Using the notation from above, n_2 is the overall number of interface nodes, whereas n_1 contains all inner nodes. n_{1j} is the number of nodes belonging to the domain j. In the context of Domain Decomposition, D_j contains the coupling from the inner nodes of one domain to each other, B_j represents the interface-to-subdomain coupling, C_j the subdomain-to-interface coupling and finally S_A the coupling of all interface nodes to each other.

Furthermore, $A_{2,2}$ is the sum of contributions from every domain, i.e. $A_{2,2} = \sum_i E_i$. The parallel application of S_A to a vector $w \in \mathbb{R}^{n_2}$, as it occurs in the (P)CG algorithm, is depicted in algorithm 6.

Algorithm 6 Application of S_A to a vector in parallel.

```
Apply $S_A$ to $w \in \mathbb{R}^{n_2}$.
for $j = 1$ to $p$ do
    Solve $D_j r_{1j} = B_j w$
    Determine $E_j w - C_j r_{1j}$
end for
$S_A w = \sum_{j=1}^{p} E_j w - C_j r_{1j}$
```

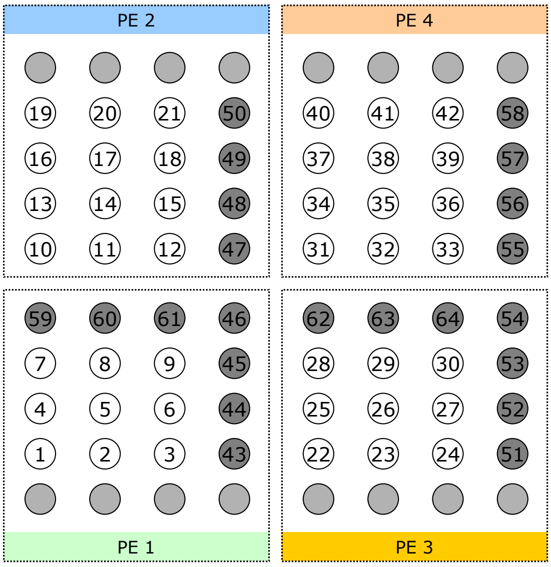

Figure 3.4: Numbering of a two-dimensional domain with Dirichlet boundary conditions in the first and periodic boundary conditions in the second direction. 4 PE's are employed. For the nodes coloured in light grey a Dirichlet condition is given. The nodes in dark grey are interface nodes.

We remark that the number of interface nodes adjacent to one domain is much smaller than n_2. In B_j and C_j, there are only entries contained corresponding to nodes on the boundary of domain j. Therefore, only information for these nodes must be transferred to PE j, which reduces the communication between the nodes significantly. In fact, in two dimensions every interface node is adjacent to only two inner domains such that the send and receive operations for this interface node in algorithm 6 must only be performed between two PE's. No global reduction operation is necessary, except for calculating the global residual.

Of course, that the boundary may be broader than one node. But in the context of 3.2, one node is sufficient since every node is only connected to its nearest neighbour. If e.g. another ansatz space for the Finite Element Method is chosen or difference stencils of higher order in a Finite Difference Method are chosen, more nodes must be contained in the boundary. Furthermore, if a Finite Difference Method is employed, A and therefore S_A is in general not symmetric and positive definite. Neither the CG algorithm nor the Cholesky decomposition can be employed.

In one or three spatial dimensions, a similar procedure leads to analogous results. In one dimension using a one-node boundary, Dirichlet boundary conditions and p PE's, there are only $p-1$ boundary nodes, which means that S_A is a $(p-1)\times(p-1)$ matrix. Thereby the effort is justifiable to determine S_A explicitly. Every PE must solve at most two linear

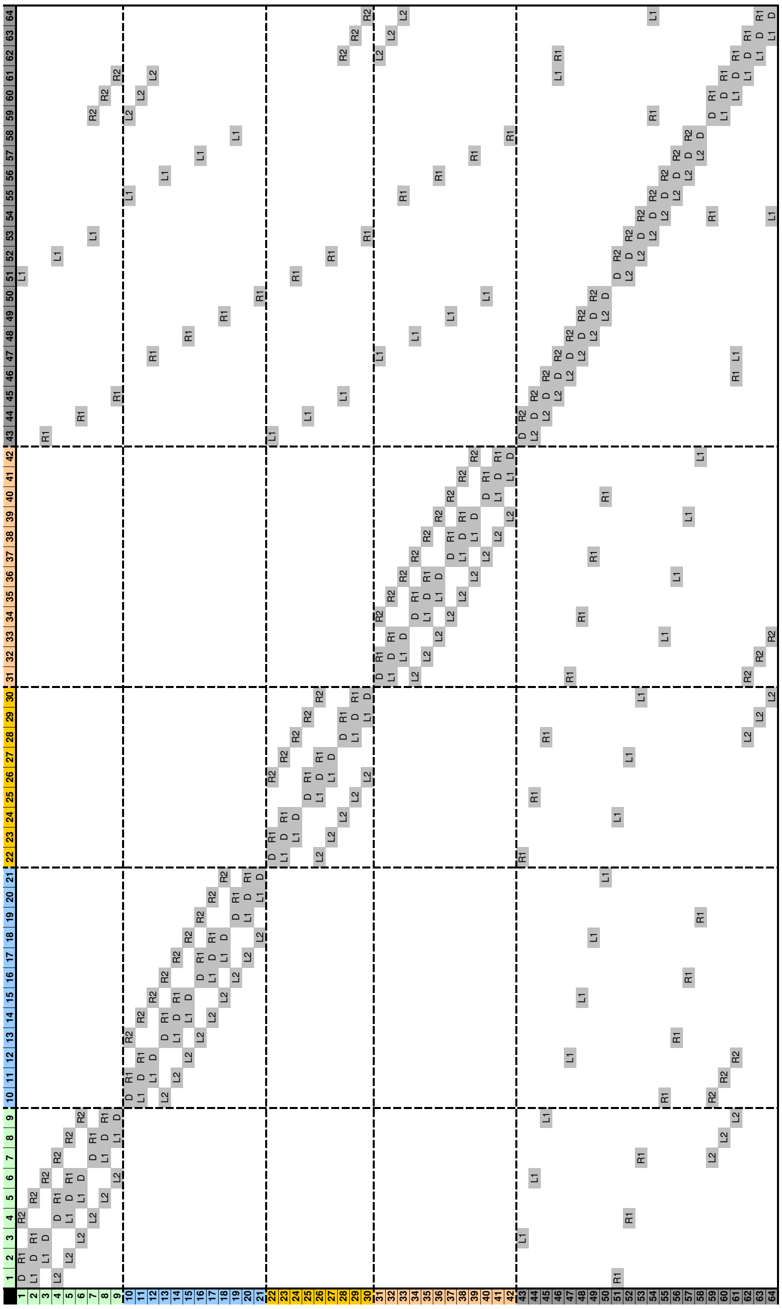

Figure 3.5: Matrix associated with the numbering of the grid in Figure 3.4.

systems, one for the upper and one for the lower boundary. The entries are gathered at one PE where the Schur Complement Equation is solved in serial mode. In fact, this might even be advisable since the system is so small that it can be solved directly.

When the CG algorithm is employed to solve the Schur Complement Equation, in every application of S_A as described in algorithm 6, the local block D_j must be inverted on every PE. Therefore, the runtime costs of algorithm 5 mainly depend on how fast the local problem can be solved repeatedly. If the Cholesky decomposition of D_j is known, this can be done very effectively, but in most cases the dimension of D_j will be too large such that the Cholesky decomposition cannot be calculated with reasonable effort. On the other hand, once a preconditioner M is determined, it can be used in every step without changes. In this case, it is justifiable to spend more time in the determination of M as required e.g. in the context of an Incomplete Cholesky Decomposition with fill-in described in 3.4.

The parallel performance as well as other interesting numerical data concerning the implementation of the methods presented in this chapter are depicted in chapter 4.

Chapter 4

Numerical Results

In this chapter, four two-dimensional test cases are presented with which the Schur Complement Method is checked. Special attention was paid to the parallel performance of the algorithm. Furthermore, some data is shown to illustrate certain issues of the numerical implementation. To solve the Schur Complement Equation, the (preconditioned) CG algorithm is used as described in section 3.5.

4.1 Test Case 1

In the first test case, Poisson's Equation $-\Delta u = f$ is solved on a rectangular and equidistant grid, with 400 nodes in each direction. In the x direction, homogeneous Dirichlet boundary conditions are set, whereas the boundary in y direction is periodic. The latitude of the computational domain is $1.5Mm \times 2.25Mm$. In all of the following pictures, the x direction is horizontal and the y direction vertical.

The equation is solved twice with right-hand sides f_1 and f_2 as in figure 4.1 resp. 4.2. In the following, these two cases are referred to by test case 1a resp. test case 1b. The corresponding solutions u_1 and u_2 are shown in figure 4.3 resp. 4.4. f_1 is constant in y direction and has a discontinuity at the lower boundary in x direction. f_2 is highly oscillating in both directions.

In table 4.1, the effect of preconditioning the CG algorithm is described. The runtimes t_1 and t_2 which are relative to the runtime without preconditioning indicate that the performance of the Schur Complement Method can be improved significantly by improving the performance of the local solver as it is done here by preconditioning. With an Incomplete Cholesky Decomposition with fill-in 8 or 10, the number of iterations in the CG algorithm can be reduced by a factor 10, similar to the results in [12]. Thereby, $n_{\text{iterations},1}$ and $n_{\text{iterations},2}$ refer to the number of iterations in the CG algorithm to calculate the right-hand side of the Schur Complement Equation. This results in a reduction of the

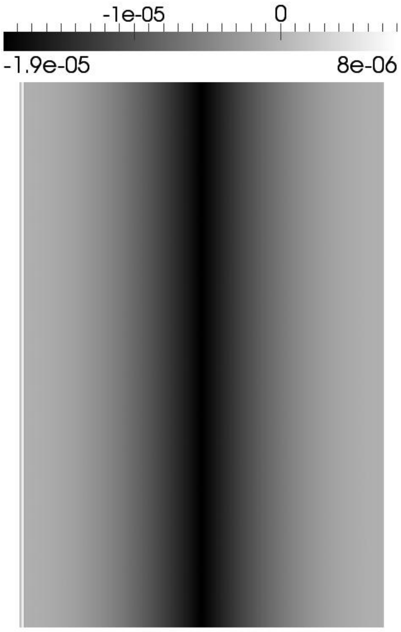

Figure 4.1: Right-hand side f_1 in test case 1a.

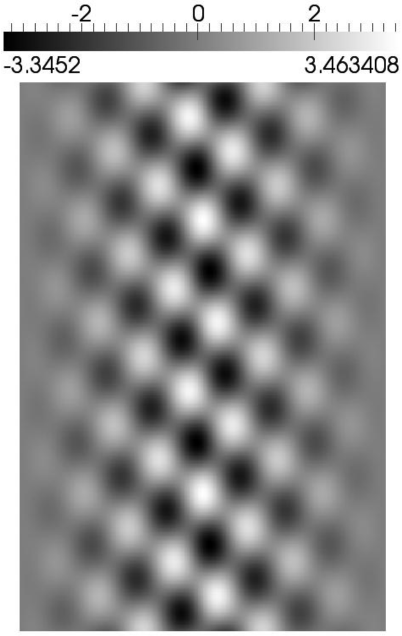

Figure 4.2: Right-hand side f_2 in test case 1b.

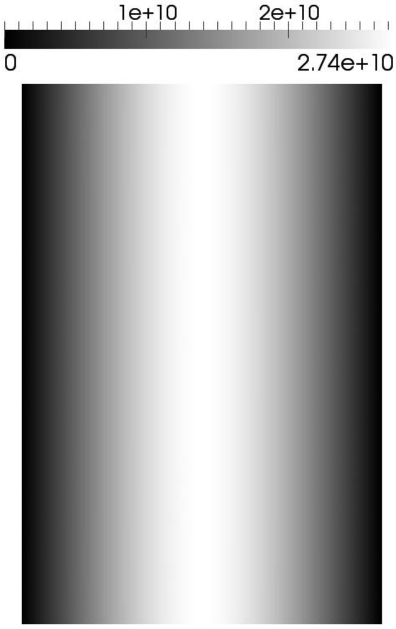

Figure 4.3: Solution u_1 in test case 1a.

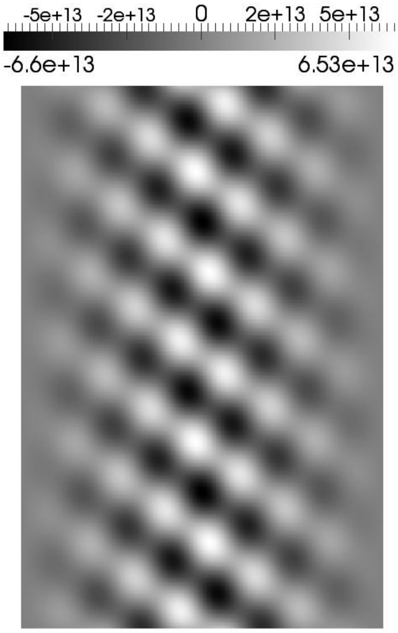

Figure 4.4: Solution u_2 in test case 1b.

Table 4.1 Influence of the choice of η for the Schur Complement Method.

η	$n_{\text{iterations},1}$	$n_{\text{iterations},2}$	t_1	t_2
0	1164	1555	1.000	1.000
2	358	507	0.497	0.418
4	241	344	0.356	0.287
6	148	213	0.285	0.215
8	121	171	0.278	0.200
10	109	141	0.279	0.193

overall runtime by a factor 4.

As stop criterion of the CG algorithm, the median of $1.0 \cdot 10^{-10}$, $1.0 \cdot 10^{-16} \cdot \langle s^{(0)}, r^{(0)} \rangle$ and $1.0 \cdot 10^{-20}$ was chosen, whereas the iteration for the Schur Complement Equation stops, if the residual is smaller than the median of $1.0 \cdot 10^{-6}$, $1.0 \cdot 10^{-16} \cdot \langle r^{(0)}, r^{(0)} \rangle$ and $1.0 \cdot 10^{-16}$, which both are quite restrictive choices.

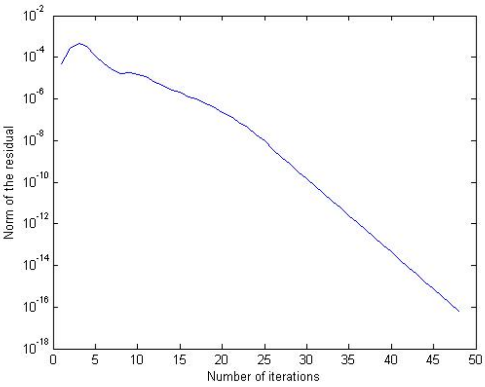

Figure 4.5: Logarithmic plot of the residual for test case 1a.

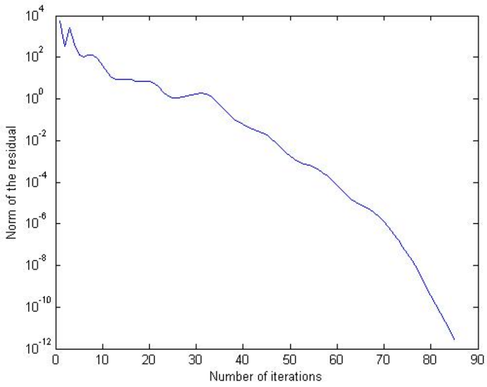

Figure 4.6: Logarithmic plot of the residual for test case 1b.

With the tool *gprofiler*, execution profiles of Fortran programs can be produced where the number of calls and the amount of time spent in each subroutine can be calculated. The result of profiling the Schur Complement Method with different values of η is presented in table 4.2. The profiled run was executed on one PE. Since the boundary in y direction is periodic, there is a "inner" boundary and the Schur Complement Method can be applied. Surely, it is generally not advisible to use this method with only one PE because this case could be covered much faster by simply using the CG algorithm.

Increasing η, the number fo fill-in of the preconditioner L, results in much less iterations of the CG algorithm, but also in more time spent in its inversion. Therefore, it is not advisible to set η larger than 8 or 10, as the data in table 4.1 indicate.

For the values of η considered, the time spent in calculating the preconditioner L is always

Table 4.2 Profiling the Schur Complement Method: breakdown by subroutines (without sub-calls).

η	overall runtime	Apply $(LL^T)^{-1}$		Apply A		$\langle\cdot,\cdot\rangle$		CG algorithm	
	in s	calls	in %	calls	in %	calls	in %	calls	in %
2	585.34	12329	50.53	12329	16.41	24802	16.27	144	14.47
4	404.31	8467	50.09	8467	16.26	17081	16.08	147	14.22
6	304.40	5398	56.42	5398	13.46	10947	13.66	151	11.92
8	284.94	4408	61.41	4408	11.75	8969	11.47	153	10.28
10	275.04	3723	65.09	3723	10.22	7601	10.29	155	8.81
4	807.20	14401	43.01	14401	28.18	28943	14.69	141	12.43

Table 4.3 Illustration of the decrease in magnitude of the entries of the Incomplete Cholesky Decomposition.

row	$(i,j-1)$	$(i+1,j-1)$	$(i+2,j-1)$	$(i+3,j-1)$	$(i+4,j-1)$
$10^{-6}\cdot$	-0.7854	-0.3720	-0.1892	-0.1024	-0.0575
row	$(i+5,j-1)$	$(i-4,j)$	$(i-3,j)$	$(i-2,j)$	$(i-1,j)$
$10^{-6}\cdot$	-0.0318	-0.0143	-0.0337	-0.0664	-1.8966

negligible. Therefore, it is not considered in table 4.2. This is not the case if η gets larger. Employing algorithm 9 instead of 8 in the CG algorithm results in longer runtimes and slower convergence due to rounding errors. The last line on table 4.2 is data from a run where algorithm 8 was employed.

In the line corresponding to the node (i,j), the lower part of the matrix A has non-zero entries in the rows corresponding to the nodes $(i,j-1)$, $(i-1,j)$ and (i,j). The additional non-zero entries in the preconditioner L with fill-in η are therefore to the right from row $(i,j-1)$ and to the left from $(i-1,j)$. They get smaller in magnitude the bigger the distance to the these rows becomes, as shown in table 4.3 where $\eta = 10$.

4.2 Test Case 2

In the second test case, κ, c and f in (2.1.10) are given by

$$\begin{aligned}
\kappa(x,y) =& \sqrt{x+1}, \\
c(x,y) =& \frac{x+y}{10}, \\
f(x,y) =& -\frac{1}{2\sqrt{x+1}} + 0.4 \cdot \sqrt{x+1} \cdot \cos(0.2 \cdot y) + \frac{x+y}{10} \cdot (x + 10 \cdot \cos(0.2 \cdot y)).
\end{aligned}$$

The grid has 240 nodes in the x and 200 nodes in the y direction. In each direction, the (constant) spacing is set to 1 and inhomogeneous Dirichlet boundary conditions are used.

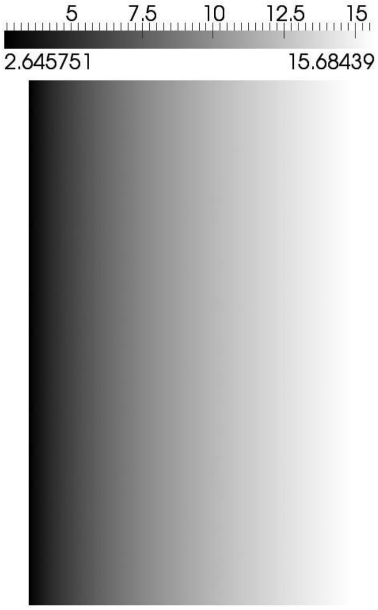

Figure 4.7: κ in test case 2.

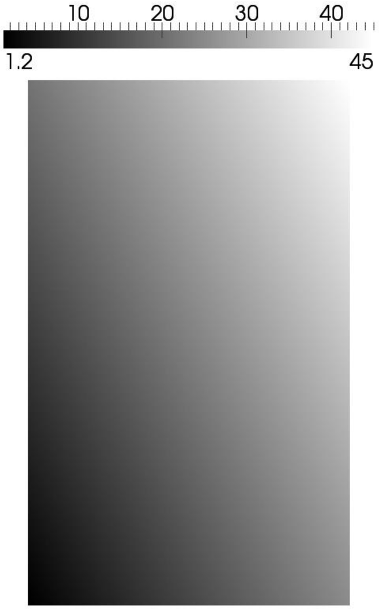

Figure 4.8: c in test case 2.

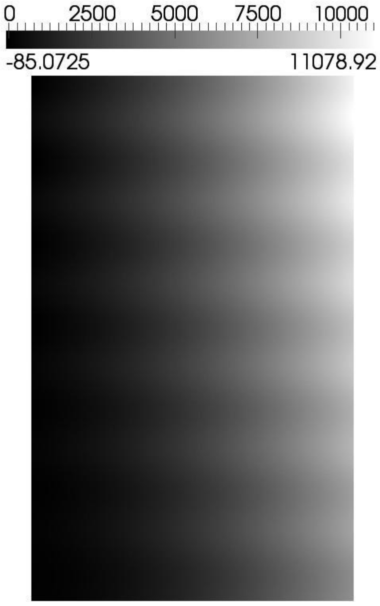

Figure 4.9: f in test case 2.

Since we know the analytical solution of this problem,

$$u_{\text{exact}}(x, y) = x + 10 \cdot \cos(0.2 \cdot y),$$

we can immediately check the correctness of the numerical results. The absolute error shown in figure 4.12 is calculated by

$$abserr(x, y) = u_{\text{exact}}(x, y) - u_{\text{num}}(x, y),$$

where u_{num} is the numerical solution given by the Schur Complement Method. The values of u_{num} considered in figure 4.12 are calculated with 4 PE's, using two subdivisions in x and two subdivisions in y direction. Compared to the magnitude of u_{num} and u_{exact}, the error is insignificant. The stop criteria in the algorithms are the same as in test case 1.

The parallel performance of the Schur Complement Method is depicted in table 4.4. When one PE is employed, $n_2 = 0$ since there is no inner boundary. Therefore the algorithm is much faster since the Schur Complement Equation must not be solved. It takes four PE's to get the same performance as in the non-parallel case. Furthermore, the data indicate that the scaling is quite well, even though the domain is small. The more PE's are used the better the performance might get. In all tests in this section, the Incomplete Cholesky Decomposition with fill-in 10 is used as a preconditioner.

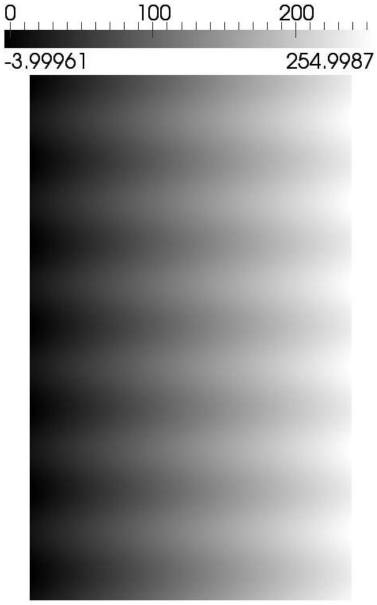

Figure 4.10: Exact solution u_{exact} in test case 2.

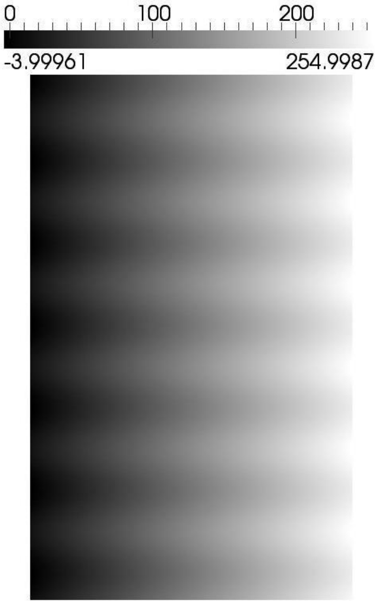

Figure 4.11: Numerical solution u_{num} in test case 2.

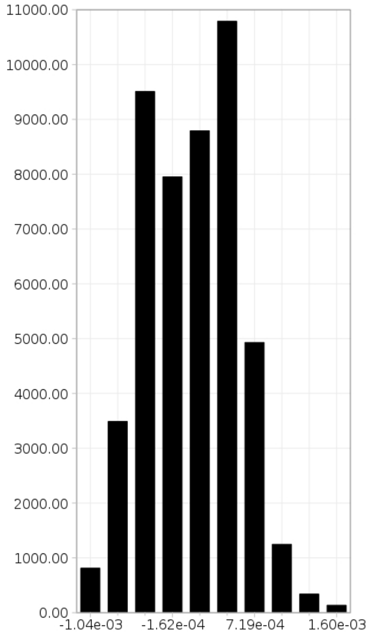

Figure 4.12: Distribution of the absolute error in test case 2.

Table 4.4 Overall runtime in dependence of the number of PE's in test case 2.

PE's	subdivisions in x dir.	subdivisions in y dir.	time spent in ms
1	1	1	867.0
2	1	2	1465.0
2	2	1	1304.0
4	1	4	880.0
4	2	2	845.0
4	4	1	851.0
8	2	4	503.0
8	4	2	442.0

4.3 Test Case 3

In the third test case, discontinuous data is considered. The domain is equidistant with constant spacing 1 and 600 nodes in x and 800 nodes in y direction. The boundary in y direction is periodic, whereas the boundary values in x direction are set to

$$u(x_1, \cdot) = 5, \ u(x_2, \cdot) = 30,$$

where x_1 and x_2 are the domain bounds in x direction. Furthermore,

$$\begin{aligned} \kappa(x, y) &= \begin{cases} x + 1, & x < \frac{x_1 + x_2}{2}, \\ 1, & \text{else}, \end{cases} \\ f(x, y) &= \begin{cases} 10, & x < \frac{x_1 + x_2}{2}, \\ x, & \text{else}, \end{cases} \\ c(x, y) &= 0. \end{aligned}$$

As figure 4.14 demonstrates, the solution is not differentiable where the coefficient functions are discontinuous.

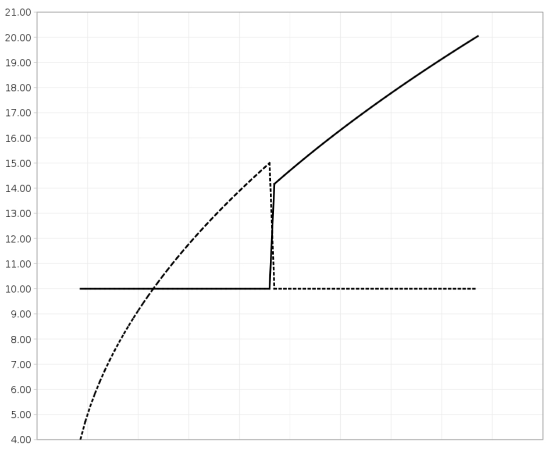

Figure 4.13: Cut along the x axis through κ (dashed) and f (solid line).

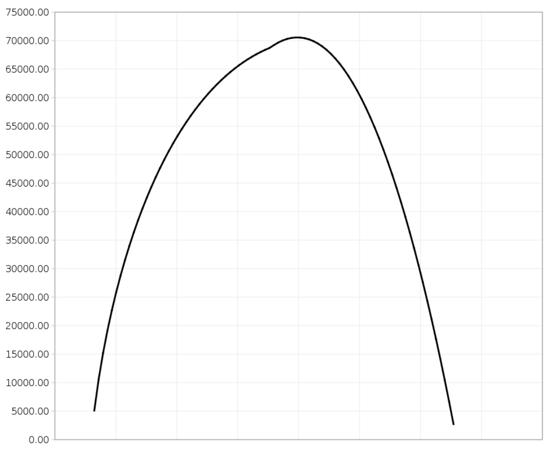

Figure 4.14: Cut along the x axis through u.

Figure 4.5 demonstrates that the scaling of the algorithm is quite good, but the method is not efficient with only few processors. When 1 PE is employed, the runtime is much shorter with the CG algorithm instead of the Schur Complement Method. In the last column, the amount of overall CPU time relative to the 1 PE case is given.

Table 4.5 Overall runtime in dependence of the number of PE's in test case 3.

PE's	subdivisions in x dir.	subdivisions in y dir.	time spent in ms	ratio
1	1	1	1510768	1.00
2	1	2	558858	0.74
2	2	1	1044287	1.38
4	2	2	336422	0.89
8	2	4	87801	0.46
8	4	2	92756	0.49
16	4	4	29538	0.31
36	6	6	11956	0.28
49	7	7	11182	0.36
64	8	8	7592	0.32

4.4 Test Case 4

In the fourth test case, to the right-hand side of

$$-\Delta u = 0,$$

with $u(x_1, \cdot) = 8,\ u(x_2, \cdot) = 0.3$, where x_1 and x_2 are the domain bounds in x direction, is disturbed with random numbers of dimension 10^{-4} to get a new right-hand side $\tilde{f}$. In y direction, periodic boundary conditions are used. Then, the equation $-\Delta u = \tilde{f}$ is solved. The numerical solution should not noticeably differ from the analytical solution of the undisturbed equation,

$$u_{\text{undisturbed}}(x, y) = \frac{7.7}{x_1 - x_2} \cdot x + \frac{0.3x_1 - 8x_2}{x_1 - x_2}.$$

In figure 4.15, the numerical solution of the disturbed problem is depicted together with the perturbation $\tilde{f}$. The perturbation has no noticeable influence on the solution, as desired.

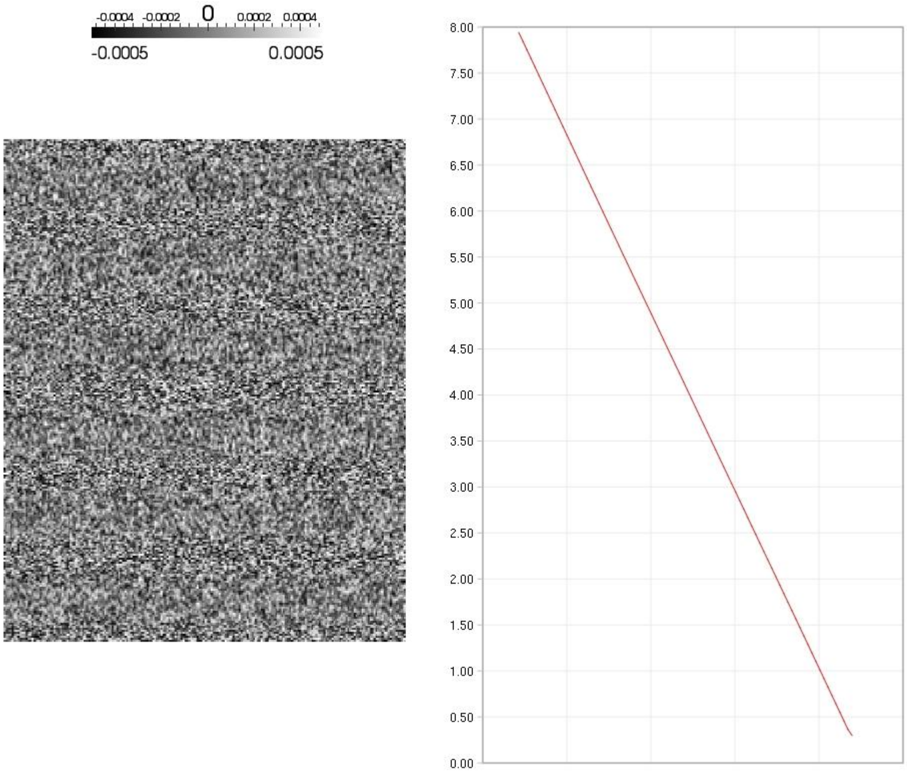

Figure 4.15: $\tilde{f}$ and cut along the x axis through u.

Chapter 5

Outlook

If the Schur Complement Equation is solved iteratively, in every iteration a local problem must be solved. Therefore, the parallel performance of the Schur Complement Equation strongly depends on the performance of the local solver. The deployment of the Incomplete Cholesky Decomposition with fill-in exploited the fact, that the time spent calculating a preconditioner is insignificant if the preconditioner can be used repeatedly.
Instead of the (preconditioned) CG algorithm, any other method to solve the local problem may be employed. For the Schur Complement Method, this does not change anything. E.g. multigrid methods or the concept of hierarchical matrices, especially in three dimensions, could be used to further improve the method.
Another possibility for future work is to find a preconditioner for the Schur matrix S_A and thereby precondition the Schur Complement Equation. Lowering the number of iterations for the iterative solution of this equation would diminish the importance of the local solver.
The tests done so far indicate that the method needs about four PE's to solve the problem as fast as it can be done by the preconditioned CG algorithm in serial mode. If the good scaling ratio is conserved even for more PE's than considered here, the method is well suited at least for systems large enough to keep many PE's busy.
Concerning the numerical grid, three assumptions were made:

(i) The spacing between two nodes is constant in every direction.

(ii) The grid is rectangular.

(iii) The grid is regular, i.e. in two dimensions, every node is connected only to its left, right, upper and lower neighbour.

These had the consequences, that the calculation of the stiffness matrix A was especially simple because every triangle (in two dimensions) of the Finite Element Method had the same shape and surface area. Furthermore, we used the regular structure of the grid which resulted in the five-diagonal form of A to develop a good preconditioner which can be easily calculated.

Omitting the first two assumptions only results in a slightly more complicated assembly of the stiffness matrix A. Therefore, polar or spherical coordinates can be included without any problems. Considering a non-regular grid results in more effort to calculate a good preconditioner, as described e.g. in [17]. In every case, due to the characteristics of the Finite Element Method, the stiffness matrix remains symmetric and positive definite and all of the numerical methods described in chapter 3 except for the preconditioning part can be employed without any changes.

Appendix A

An Implementation of the Finite Element Method

On the next pages, an implementation of the Finite Element Method applied to (2.1.10) with various boundary conditions is presented. The domain Ω is a straight line, a rectangle or a cuboid, depending on the dimension of space. Ω is subdivided by an equidistant grid in each direction. As ansatz space, I choose the space of linear splines and its analogues in higher dimensions. As already mentioned, more complicated geometries are possible, but will not be considered in the following. Again, $\kappa, c \in L^\infty(\Omega)$ such that $0 < \kappa_0 \leq \kappa(x) \leq \kappa_\infty$ and $0 \leq c(x) \leq c_\infty$. Furthermore, $f \in L^2(\Omega)$.

A.1 On a Straight line

If $\Omega = [x_1, x_n]$, where $x_1, x_n \in \mathbb{R}$, then (2.1.10) with homogeneous Dirichlet boundary conditions becomes

$$-\frac{d}{dx}\left(\kappa(x)\frac{d}{dx}u(x)\right) + c(x)u(x) = f(x),\ x \in [x_1, x_n],\ u(x_1) = u(x_n) = 0, \tag{A.1.1}$$

which is a Sturm-Liouville operator. Let the grid be given by

$$\{x_i : i = 1, \ldots, n\} \text{ with } x_{i+1} = x_i + h_x, \tag{A.1.2}$$

where $h_x \in \mathbb{R}$ is the constant spacing between two grid points. We write $' = \frac{d}{dx}$. The functions Λ_i given by

$$\Lambda_i(x) = \begin{cases} \frac{x-x_{i-1}}{h_x}, & x_{i-1} \leq x \leq x_i, \\ \frac{x_i-x}{h_x}, & x_i \leq x \leq x_{i+1}, \\ 0, & \text{else}, \end{cases}, \ i = 1, \ldots, n, \tag{A.1.3}$$

$$\Lambda_i'(x) = \begin{cases} \frac{1}{h_x}, & x_{i-1} < x < x_i, \\ -\frac{1}{h_x}, & x_i < x < x_{i+1}, \\ 0, & \text{else}, \end{cases}, \ i = 1, \ldots, n, \tag{A.1.4}$$

form a basis of the space of linear splines on Ω. The functions Λ_i are often called "hat functions". These functions are such that $\Lambda_i(x_j) = \delta_{ij}$, where δ_{ij} is the Kronecker symbol, and they are affine on each subinterval (x_i, x_{i+1}) of Ω.

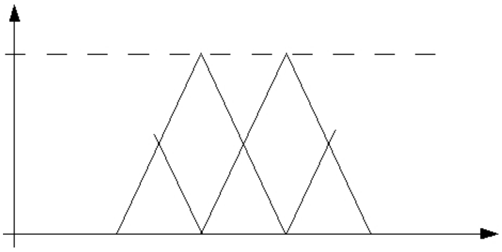

Figure A.1: The hat functions $\Lambda_{i,j}$.

Now we can determine the entries of A and b defined by (3.2.5):

$$\begin{aligned} a_{i,i} = a(\Lambda_i, \Lambda_i) &= \int_{x_{i-1}}^{x_{i+1}} \kappa(x)(\Lambda_i'(x))^2 dx + \int_{x_{i-1}}^{x_{i+1}} c(x)\Lambda_i(x)^2 dx \\ &= \frac{1}{h_x^2} \int_{x_{i-1}}^{x_{i+1}} \kappa(x) dx + \int_{x_{i-1}}^{x_{i+1}} c(x)\Lambda_i(x)^2 dx, \\ a_{i,i+1} = a(\Lambda_i, \Lambda_{i+1}) &= \int_{x_i}^{x_{i+1}} \kappa(x)\Lambda_i'(x)\Lambda_{i+1}'(x) dx + \int_{x_i}^{x_{i+1}} c(x)\Lambda_i(x)\Lambda_{i+1}(x) dx \\ &= -\frac{1}{h_x^2} \int_{x_i}^{x_{i+1}} \kappa(x) dx + \int_{x_i}^{x_{i+1}} c(x)\Lambda_i(x)\Lambda_{i+1}(x) dx, \\ b_i = l(\Lambda_i) &= \int_{x_{i-1}}^{x_{i+1}} f(x)\Lambda_i(x) dx. \end{aligned}$$

All other entries of A are 0 since the support of the associated hat functions does not overlap. To evaluate the integrals, one can use the **trapezoidal rule**

$$\int_{x_i}^{x_{i+1}} f(x)\, dx \approx \frac{h_x}{2}(f(x_i) + f(x_{i+1})). \tag{A.1.5}$$

Applying this to the integrals from above gives

$$\begin{aligned}
\int_{x_{i-1}}^{x_{i+1}} \kappa(x)dx &= \int_{x_{i-1}}^{x_i} \kappa(x)dx + \int_{x_i}^{x_{i+1}} \kappa(x)dx \\
&\approx \frac{h_x}{2}(\kappa_{i-1} + \kappa_i) + \frac{h_x}{2}(\kappa_i + \kappa_{i+1}), \\
\int_{x_{i-1}}^{x_{i+1}} c(x)\Lambda_i(x)^2 dx &= \int_{x_{i-1}}^{x_i} c(x)\Lambda_i(x)^2 dx + \int_{x_i}^{x_{i+1}} c(x)\Lambda_i(x)^2 dx \\
&\approx \frac{h_x}{2}(c_{i-1} \cdot 0 + c_i \cdot 1) + \frac{h_x}{2}(c_i \cdot 1 + c_{i+1} \cdot 0) = h_x \cdot c_i, \\
\int_{x_i}^{x_{i+1}} \kappa(x)dx &\approx \frac{h_x}{2}(\kappa_i + \kappa_{i+1}), \\
\int_{x_i}^{x_{i+1}} c(x)\Lambda_i(x)\Lambda_{i+1} dx &\approx \frac{h_x}{2}(c_i \cdot 1 \cdot 0 + c_{i+1} \cdot 0 \cdot 1) = 0, \\
\int_{x_{i-1}}^{x_{i+1}} f(x)\Lambda_i(x)dx &= \int_{x_{i-1}}^{x_i} f(x)\Lambda_i(x)dx + \int_{x_i}^{x_{i+1}} f(x)\Lambda_i(x)dx \\
&\approx \frac{h_x}{2}(f_{i-1} \cdot 0 + f_i \cdot 1) + \frac{h_x}{2}(f_i \cdot 1 + f_{i+1} \cdot 0) = h_x \cdot f_i,
\end{aligned}$$

where $\kappa_i = \kappa(x_i)$ and so on. Therefore, the system $A\mathbf{u}_h = b$ with homogeneous Dirichlet boundary conditions in x_1 and x_n, i.e. $u_1 = u_n = 0$, has the form

$$\begin{pmatrix} a_{2,2} & a_{2,3} & 0 & \cdots \\ a_{2,3} & a_{3,3} & a_{3,4} & \cdots \\ 0 & a_{3,4} & a_{4,4} & \cdots \\ \vdots & & \ddots & \\ \cdots & 0 & a_{n-2,n-1} & a_{n-1,n-1} \end{pmatrix} \begin{pmatrix} u_2 \\ u_3 \\ u_4 \\ \vdots \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_{n-1} \end{pmatrix}, \tag{A.1.6}$$

where

$$\begin{aligned}
a_{i,i} &= \frac{1}{2h_x}(\kappa_{i-1} + 2 \cdot \kappa_i + \kappa_{i+1}) + h_x \cdot c_i, \\
a_{i,i+1} &= -\frac{1}{2h_x}(\kappa_i + \kappa_{i+1}), \\
b_i &= h_x \cdot f_i.
\end{aligned}$$

This implies that κ as well as all other variables are given on the nodes x_i. Usually in the context of hydrodynamic codes, some variables are given on the cell boundaries, which in the 1D case correspond to the points $x_{i+\frac{1}{2}}$. Then, the integration presented above must be changed. E.g., one could first linearly interpolate the values on the cell center from the boundary values and then apply the trapezoidal rule on the central grid, or one could instead use the **midpoint rule**

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \approx h_x \cdot f(x_{i+\frac{1}{2}}).$$

Algorithm 7 Calculation of the entries of $A = [a_{i,j}]$.

```
for $i = 1$ to $n$ do
    if $\kappa$ is given on the central grid then
        $a_{i,i-1} = -\frac{1}{2h_x}(\kappa_i + \kappa_{i-1})$
        $a_{i,i} = \frac{1}{2h_x}(\kappa_{i-1} + 2 \cdot \kappa_i + \kappa_{i+1}) + h_x \cdot c_i$
        $a_{i,i+1} = -\frac{1}{2h_x}(\kappa_i + \kappa_{i+1})$
    else if $\kappa$ is given on the boundary grid and the midpoint rule is applied then
        $a_{i,i-1} = -\frac{1}{h_x}\kappa_{i-\frac{1}{2}}$
        $a_{i,i} = \frac{1}{h_x}(\kappa_{i-\frac{1}{2}} + \kappa_{i+\frac{1}{2}}) + h_x \cdot c_i$
        $a_{i,i+1} = -\frac{1}{h_x}\kappa_{i+\frac{1}{2}}$
    end if
end for
```

Considering inhomogeneous Dirichlet boundary conditions $u_1 = g_1$ and $u_n = g_n$ for given $g_1, g_n \in \mathbb{R}$, we see that

$$u_0 : \Omega \to \mathbb{R}, u_0(x) = \begin{cases} g_1 \cdot \frac{x_2 - x}{h_x}, & x \leq x_2, \\ g_n \cdot \frac{x - x_{n-1}}{h_x}, & x \geq x_{n-1}, , \\ 0, & \text{else}, \end{cases} \tag{A.1.7}$$

fulfils the requirements described in Section 3.2. Now by

$$\begin{aligned} a(u_0, \Lambda_i) &= \int_{x_{i-1}}^{x_{i+1}} \kappa(x) u_0'(x) \Lambda_i'(x) dx + \int_{x_{i-1}}^{x_{i+1}} c(x) u_0(x) \Lambda_i(x) dx \\ &\approx \begin{cases} -\frac{1}{2h_x}(\kappa_{i-1} + \kappa_i), & i = 2, \\ -\frac{1}{2h_x}(\kappa_i + \kappa_{i+1}), & i = n-1, \\ 0, & \text{else}, \end{cases} \end{aligned}$$

it follows that if b_2 and b_{n-1} are changed accordingly and u_1 and u_n are set to the desired boundary values, $u_h \in H^1(\Omega)$ solves the inhomogeneous problem.

If Neumann boundary conditions $u'(x_1) = g_1$ and $u'(x_n) = g_n$ are given, the equations for u_1 and u_n must be added to the matrix since these values are now unknown. The right-hand side is changed accordingly.

$$\left(\frac{1}{2h_x}(\kappa_1+\kappa_2)+\frac{h_x}{2}c_i\right)\cdot u_1-\frac{1}{2h_x}(\kappa_1+\kappa_2)\cdot u_2=\frac{h_x}{2}f_1-\kappa_1 g_1$$
$$\left(\frac{1}{2h_x}(\kappa_{n-1}+\kappa_n)+\frac{h_x}{2}c_n\right)\cdot u_n-\frac{1}{2h_x}(\kappa_{n-1}+\kappa_n)\cdot u_{n-1}=\frac{h_x}{2}f_n-\kappa_n g_n.$$

If $c=0$, the condition (2.1.23) tranforms to

$$\int_\Omega f(x)dx=\kappa_1 g_1-\kappa_n g_n.$$

Therefore, given a Neumann boundary condition on one side, the other side is also fixed, but still there are many solutions which differ by a constant.
Solving the linear system (A.1.6) can be done directly using its Cholesky decomposition. Since the bandwidth of A from (A.1.6) is only 3, the lower triangle matrix $L=[l_{i,j}]$ from the Cholesky decomposition of A has only two diagonals with non-zero entries. They can be determined by

$$\begin{aligned} l_{1,1}=&\sqrt{a_{1,1}},\\ l_{i+1,i}=&\frac{a_{i+1,i}}{l_{i,i}},\ i=1,\dots,n-1,\\ l_{i,i}=&\sqrt{a_{i,i}-l_{i,i-1}^2},\ i=2,\dots,n. \end{aligned}$$

A.2 On a Rectangle

Let Ω be a rectangle, i.e. $\exists x_1,x_2,y_1,y_2\in\mathbb{R}$ such that $\Omega=[x_1,x_2]\times[y_1,y_2]\subset\mathbb{R}^2$, with homogeneous Dirichlet boundary conditions in both directions.
Let the grid be given by

$$\{(x_i,y_j):i=1,\dots,n,\ j=1,\dots,m\}\text{ with }x_{i+1}=x_i+h_x,\ y_{j+1}=y_j+h_y,\tag{A.2.1}$$

where $h_x,h_y\in\mathbb{R}$ are the constant spacings between two grid points in x respective y direction. For each point $(x_i,y_j)\in\Omega$, we want to define $\Lambda_{i,j}$ such that

$$\Lambda_{i,j}(x_k,y_l)=\delta_{(i,j),(k,l)}:=\begin{cases}1, & i=k\text{ and }j=l,\\ 0, & \text{else},\end{cases}\tag{A.2.2}$$

and $\Lambda_{i,j}$ decays linearly in between. For this, the grid must be divided into triangles, which can easily be done by splitting every rectangle $[x_i, x_{i+1}] \times [y_j, y_{j+1}]$ along the first diagonal.

Equivalently, the rectangles could be split along the second diagonal, which can result in quite different results. To remedy this drawback, one could execute the following procedure for both possibilities and in the end, average the results. This is not necessary, if the value of the variables do not change significantly between two cells.

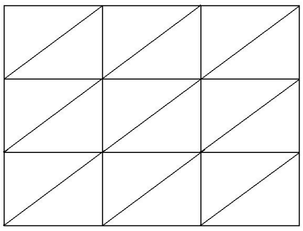

Figure A.2: Triangulation of a rectangular grid.

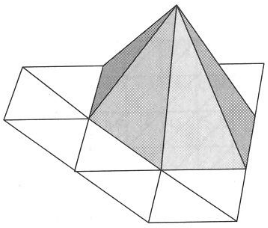

Figure A.3: Hat function $\Lambda_{i,j}$ (from [1]).

Therefore, the functions $\Lambda_{i,j}$ are of the form

$$\Lambda_{i,j}(x, y) = \beta + \alpha_1 x + \alpha_2 y, \tag{A.2.3}$$

where $\beta, \alpha_1, \alpha_2$ are different on every triangle T_k and $\nabla \Lambda_{i,j}|_{T_k} = (\alpha_1, \alpha_2)^T$. The support of $\Lambda_{i,j}$ consists of the six triangles adjacent to (x_i, y_j).

Now we can determine the entries of $A = [a_{(i,j),(k,l)}]$ and $b = [b_{i,j}]$:

$$\begin{aligned} a_{(i,j),(i,j)} = a(\Lambda_{i,j}, \Lambda_{i,j}) &= \int_\Omega \kappa(x, y) \left|\nabla \Lambda_{i,j}(x, y)\right|^2 d(x, y) + \int_\Omega c(x, y) \left|\Lambda_{i,j}(x, y)\right|^2 d(x, y) \\ &= \sum_{k=1}^{6} \int_{T_k} \kappa(x, y) \left|\nabla \Lambda_{i,j}(x, y)\right|^2 + c(x, y) \left|\Lambda_{i,j}(x, y)\right|^2 d(x, y) \\ b_{(i,j)} = l(\Lambda_{i,j}) &= \int_\Omega f(x, y) \Lambda_{i,j}(x, y) d(x, y) = \sum_{k=1}^{6} \int_{T_k} f(x, y) \Lambda_{i,j}(x, y) d(x, y). \end{aligned}$$

There are six cases where the intersection of the support of two different nodes is non-empty.

(i) Nodes (x_i, y_j) and (x_{i+1}, y_j),

(ii) nodes (x_i, y_j) and (x_i, y_{j+1}),

(iii) nodes (x_i, y_j) and (x_{i+1}, y_{j+1}),

(iv) nodes (x_i, y_j) and (x_{i-1}, y_j),

(v) nodes (x_i, y_j) and (x_i, y_{j-1}),

(vi) nodes (x_i, y_j) and (x_{i-1}, y_{j-1}).

There are always two triangles contained in the intersection. Therefore, if (x_k, y_l) is a neighbouring point of (x_i, y_j), then

$$\begin{aligned} a_{(i,j),(k,l)} = a(\Lambda_{i,j}, \Lambda_{k,l}) &= \int_\Omega \kappa(x,y)\nabla\Lambda_{i,j}(x,y) \cdot \nabla\Lambda_{k,l}(x,y)d(x,y) \\ &+ \int_\Omega c(x,y)\Lambda_{i,j}(x,y)\Lambda_{k,l}(x,y)d(x,y) \\ &= \sum_{k=1}^{2} \int_{T_k} \kappa(x,y)\nabla\Lambda_{i,j}(x,y) \cdot \nabla\Lambda_{k,l}(x,y) \\ &+ c(x,y)\Lambda_{i,j}(x,y)\Lambda_{k,l}(x,y)d(x,y). \end{aligned}$$

To evaluate the integrals, we use the formula

$$\int_T f(x,y)d(x,y) \approx \frac{h_x h_y}{6}(f_1 + f_2 + f_3), \tag{A.2.4}$$

where T is a rectangular triangle with side length h_x and h_y and f is a real-valued function on T with f_1 denoting the value of f in the first edge of T and so on.

(A.2.4) is the two-dimensional analogue to (A.1.5). Suppose f is linear on T, i.e. $f(x,y) = f_1 + \frac{f_2 - f_1}{h_x}x + \frac{f_3 - f_1}{h_y}y$, then

$$\begin{aligned} \int_T f(x,y)\ d(x,y) &= \int_0^{h_x} \int_0^{h_y - \frac{h_y}{h_x}x} f_1 + \frac{f_2 - f_1}{h_x}x + \frac{f_3 - f_1}{h_y}y\ dy\ dx \\ &= \int_0^{h_x} \left[f_1 y + \frac{f_2 - f_1}{h_x}xy + \frac{f_3 - f_1}{h_x}\frac{y^2}{2} \right] \Big|_0^{h_y - \frac{h_y}{h_x}x}\ dx \\ &= \int_0^{h_x} f_1(h_y - \frac{h_y}{h_x}x) + \frac{f_2 - f_1}{h_x}x(h_y - \frac{h_y}{h_x}x) + \frac{f_3 - f_1}{h_y}\frac{(hy - \frac{h_y}{h_x}x)^2}{2}\ dx \\ &= \left[f_1(h_y x - \frac{h_y}{h_x}\frac{x^2}{2}) + \frac{f_2 - f_1}{h_x}(h_y\frac{x^2}{2} - \frac{h_y}{h_x}\frac{x^3}{3}) - \frac{f_3 - f_1}{h_y}\frac{(h_y - \frac{h_y}{h_x}x)^3}{6\frac{h_y}{h_x}} \right]_0^{h_x} \\ &= \frac{h_x h_y}{6}(f_1 + f_2 + f_3). \end{aligned}$$

Therefore, for linear functions the formula (A.2.4) is exact.

Using (A.2.4), the integrals can be evaluated numerically and the matrix A can be determined. Using the mapping

$$r : \{1, \dots, n\} \times \{1, \dots, m\} \to \mathbb{N},\ r(i, j) = (j - 1)n + i, \tag{A.2.5}$$

we can draw the matrix A as an element of $\mathbb{R}^{nm \times nm}$. The $(j-1)n+i$th equation is of the form:

$$a_{(i,j),(i,j-1)}u_{i,j-1} + a_{(i,j),(i-1,j)}u_{i-1,j} + a_{(i,j),(i,j)}u_{i,j} + a_{(i,j),(i+1,j)}u_{i+1,j} + a_{(i,j),(i,j+1)}u_{i,j+1} = b_{i,j}. \tag{A.2.6}$$

A short calculation shows that all entries of the form $a_{(i,j),(i+1,j+1)}$ and $a_{(i,j),(i-1,j-1)}$ are 0 since the gradients of the corresponding hat functions are orthogonal. Therefore, in every line of A there are five non-zero entries, but the bandwidth of A is larger than n. Now, calculating a Cholesky factorisation is much more expensive concerning computation time and memory requirements. To solve these systems, it is advisible to use iterative algorithms which are described in Section 3.3.

Since the matrix A is a sparse matrix, it is recommendable to store only the non-zero entries of A. This can be done by using five one-dimensional arrays, which correspond to the upper, lower, left and right neighbour of each node and to the node itself.

In the case of Poisson's Equation, i.e. $\kappa(x) = 1$, $c(x) = 0$, the matrix A has the form

$$\begin{pmatrix}
\frac{2h_x}{h_y} + \frac{2h_y}{h_x} & -\frac{h_y}{h_x} & \cdots & -\frac{h_x}{h_y} & & \cdots & & & 0 \\
-\frac{h_y}{h_x} & \frac{2h_x}{h_y} + \frac{2h_y}{h_x} & -\frac{h_y}{h_x} & \cdots & -\frac{h_x}{h_y} & \cdots & & & \\
\vdots & & \ddots & & & \ddots & & & \\
\cdots & -\frac{h_x}{h_y} & \cdots & -\frac{h_y}{h_x} & \frac{2h_x}{h_y} + \frac{2h_y}{h_x} & -\frac{h_y}{h_x} & \cdots & -\frac{h_x}{h_y} & \cdots \\
\vdots & & & \ddots & & & \ddots & & \\
0 & & & & \cdots & -\frac{h_x}{h_y} & \cdots & -\frac{h_y}{h_x} & \frac{2h_x}{h_y} + \frac{2h_y}{h_x}
\end{pmatrix}, \tag{A.2.7}$$

which has, after dividing by $h_x \cdot h_y$, the same entries as the matrix which results when the Finite Difference Method is employed to discretise Poisson's Equation. If the CG algorithm 1 is employed to solve this system, most of the computation time is spent calculating the matrix-vector product $Ad^{(k)}$. For Poisson's Equation, this can be implemented much faster since the entries of A are the same along the diagonals and they do not need to be stored and accessed separately. The effect of using algorithm 9 instead of 8 is shown in the first test case in chapter 4.

Again, if κ (or any other variable) is not given on the central, but on the boundary grid, some sort of interpolation must be performed.

Algorithm 8 Application of A in the general case.

```
Apply $A$ to $d \in \mathbb{R}^{n \times m}$.
for $i = 1$ to $n$ do
    for $j = 1$ to $m$ do
        $Ad(i,j) = a_{(i,j),(i,j)} d_{i,j}$
        $Ad(i,j) = Ad(i,j) + a_{(i,j),(i-1,j)} d_{i-1,j} + a_{(i,j),(i+1,j)} d_{i+1,j}$
        $Ad(i,j) = Ad(i,j) + a_{(i,j),(i,j-1)} d_{i,j-1} + a_{(i,j),(i,j+1)} d_{i,j+1}$
    end for
end for
```

Algorithm 9 Application of A for Poisson's Equation.

```
Apply $A$ to $d \in \mathbb{R}^{n \times m}$.
for $i = 1$ to $n$ do
    for $j = 1$ to $m$ do
        $Ad(i,j) = \left(\frac{2h_x}{h_y} + \frac{2h_y}{h_x}\right) d_{i,j}$
        $Ad(i,j) = Ad(i,j) - \frac{h_y}{h_x}(d_{i-1,j} + d_{i+1,j})$
        $Ad(i,j) = Ad(i,j) - \frac{h_x}{h_y}(d_{i,j-1} + d_{i,j+1})$
    end for
end for
```

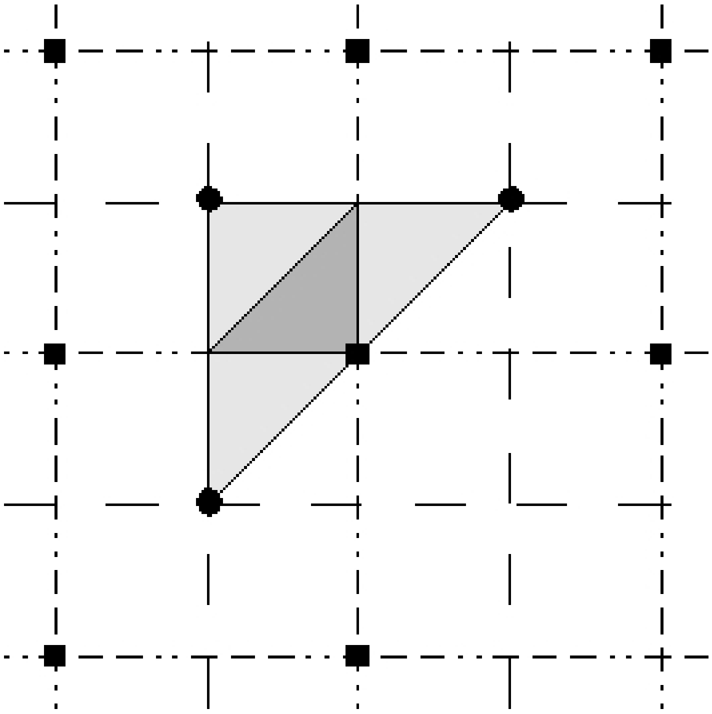

Figure A.4: The dashed line is the central grid, the big dots are nodes of the central grid. Nodes on the boundary grid (dashed-dotted) are squares.

In figure A.4, the triangle on the central grid where the integration should be performed, is in light grey. If the values of the function f that should be integrated are available only on the boundary grid, two possibilities for numerical integration are available:

(i) Suppose f is linear. Calculate the value of f in the barycentre of the triangle in darker grey by linear interpolating the values on the edges which are not on the boundary grid. The barycentre of the two triangles coincide. $\int_T f(x,y)d(x,y) = f_{\text{barycentre}} \cdot \frac{h_x h_y}{2}$.

(ii) Use a linear two-dimensional interpolation to determine the values of f on the edges of the triangle in lighter grey and then use (A.3.3).

If inhomogeneous Dirichlet boundary conditions are used, the procedure is the same as it was described before for one dimension. The right-hand side of $A\mathbf{u}_h = b$ changes to

$$\begin{aligned} b_{2,j} &= b_{2,j} - a_{(2,j),(1,j)} u_{1,j}, \\ b_{n-1,j} &= b_{n-1,j} - a_{(n-1,j),(n,j)} u_{n,j}, \ j = 1, \ldots, m, \end{aligned}$$

if the boundary is in x direction, and analogously if it is in y direction. If there is a Neumann boundary condition e.g. in $(x_1, \cdot)$ and $(x_n, \cdot)$, the right-hand side changes to

$$\begin{aligned} b_{1,j} &= b_{1,j} + h_y \kappa_{1,j} g_{1,j}, \\ b_{n,j} &= b_{n,j} + h_y \kappa_{n,j} g_{n,j}, \ j = 1, \ldots, m. \end{aligned}$$

Periodic boundary conditions change the bandwidth of A, since now – if the periodic boundary is in y direction – the nodes $(\cdot, y_1)$ and $(\cdot, y_n)$ are connected.

A.3 On a Cuboid

Let Ω be a cuboid, i.e. $\exists x_1, x_2, y_1, y_2, z_1, z_2 \in \mathbb{R}$ such that $\Omega = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2] \subset \mathbb{R}^3$, with homogeneous Dirichlet boundary conditions in all directions.
Let the grid be given by

$$\{(x_i, y_j, z_k) : i = 1, \ldots, n, \ j = 1, \ldots, m, \ k = 1, \ldots, p\}, \tag{A.3.1}$$

with $x_{i+1} = x_i + h_x, \ y_{j+1} = y_j + h_y, \ z_{k+1} = z_k + h_z$, where $h_x, h_y, h_z \in \mathbb{R}$ are the constant spacings between two grid points in the corresponding direction. For each point

$(x_i, y_j, z_k) \in \Omega$, we want to define $\Lambda_{i,j,k}$ such that

$$\Lambda_{i,j,k}(x_l, y_m, z_n) = \delta_{(i,j,k),(l,m,n)} := \begin{cases} 1, & i = l, j = m \text{ and } k = n, \\ 0, & \text{else}, \end{cases} \tag{A.3.2}$$

and $\Lambda_{i,j,k}$ decays linearly in between.

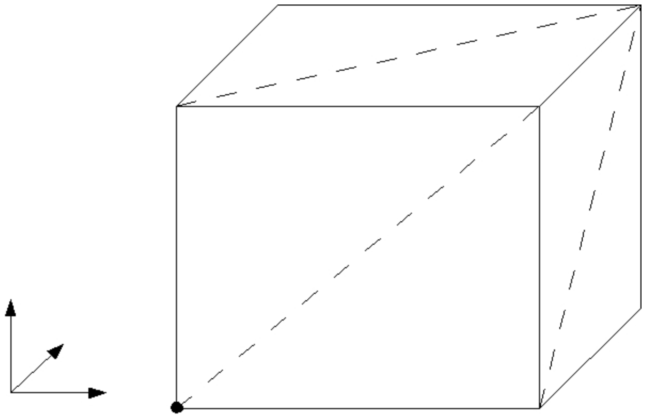

Figure A.5: Dividing a cuboid in six pyramides.

If the cuboid is divided in six pyramides as shown in figure A.5, the functions $\Lambda_{i,j,k}$ can be designed such that they are linear on each pyramid.

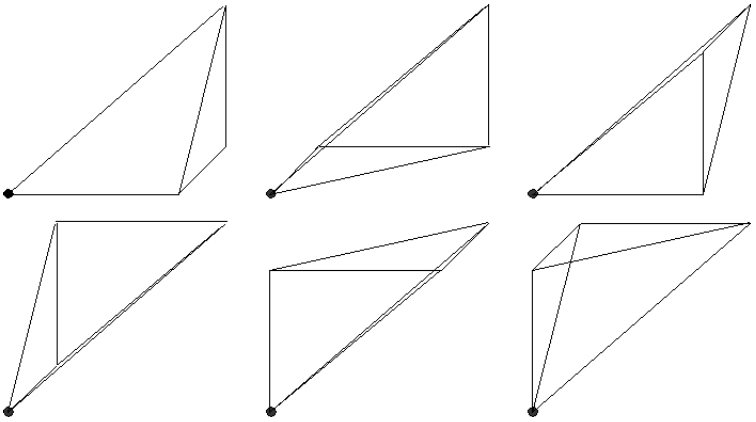

Figure A.6: The six pyramides which together build a cuboid.

If the big dot at the edge of each pyramid in Figure A.6 is the node (x_i, y_j, z_k), we can assign uniquely the tuple (i, j, k, l) to each pyramid. i, j, k are the indices of the node marked by the big dot and $l \in \{1, \ldots, 6\}$ denotes which of the six pyramides from A.6 is appealed to.
The volume of each pyramide is $\frac{h_x h_y h_z}{6}$. Analogously to before, the formula

$$\int_P f(x, y, z) d(x, y, z) \approx \frac{h_x h_y h_z}{6} \cdot \frac{f_1 + f_2 + f_3 + f_4}{4} \tag{A.3.3}$$

is applied, where f is a real-valued function on the pyramide P with values f_1 on the first edge of P and so on. Again, the formula (A.3.3) is exact for linear functions.
To simplify further calculations, we define the function V_f by

$$\begin{aligned} V_f : \{1,\dots,n\} \times \{1,\dots,m\} \times \{1,\dots,p\} \times \{1,\dots,6\} &\to \mathbb{R}, \\ V_f(i,j,k,l) &= \frac{h_x h_y h_z}{6} \cdot \frac{f_1 + f_2 + f_3 + f_4}{4}, \end{aligned}$$

using the same notation as before. $V_f(i,j,k,l)$ is the approximate value of the integral of the function f over the pyramide $P_{i,j,k,l}$.
In the same manner as before the entries of A and b are determined. Since there are no big differences to the two-dimensional case, only the results are stated. First the diagonal entries of A,

$$\begin{aligned} a_{(i,j,k),(i,j,k)} = {} & (\frac{1}{h_y^2} + \frac{1}{h_z^2}) \cdot V_\kappa(i-1,j-1,k,1) + (\frac{1}{h_x^2} + \frac{1}{h_z^2}) \cdot V_\kappa(i-1,j-1,k,2) \\ & + (\frac{1}{h_x^2} + \frac{1}{h_y^2}) \cdot V_\kappa(i-1,j,k,1) + (\frac{1}{h_x^2} + \frac{1}{h_z^2}) \cdot V_\kappa(i-1,j,k,3) \\ & + \frac{1}{h_x^2} \cdot V_\kappa(i,j,k,1) + \frac{1}{h_y^2} \cdot V_\kappa(i,j,k,2) + \frac{1}{h_x^2} \cdot V_\kappa(i,j,k,3) \\ & + \frac{1}{h_y^2} \cdot V_\kappa(i,j,k,4) + \frac{1}{h_z^2} \cdot V_\kappa(i,j,k,5) + \frac{1}{h_z^2} \cdot V_\kappa(i,j,k,6) \\ & + (\frac{1}{h_x^2} + \frac{1}{h_y^2}) \cdot V_\kappa(i,j-1,k,2) + (\frac{1}{h_y^2} + \frac{1}{h_z^2}) \cdot V_\kappa(i,j-1,k,4) \\ & + \frac{1}{h_z^2} \cdot V_\kappa(i-1,j-1,k-1,1) + \frac{1}{h_z^2} \cdot V_\kappa(i-1,j-1,k-1,2) \\ & + \frac{1}{h_y^2} \cdot V_\kappa(i-1,j-1,k-1,3) + \frac{1}{h_x^2} \cdot V_\kappa(i-1,j-1,k-1,4) \\ & + \frac{1}{h_y^2} \cdot V_\kappa(i-1,j-1,k-1,5) + \frac{1}{h_x^2} \cdot V_\kappa(i-1,j-1,k-1,6) \\ & + (\frac{1}{h_y^2} + \frac{1}{h_z^2}) \cdot V_\kappa(i-1,j,k-1,3) + (\frac{1}{h_x^2} + \frac{1}{h_y^2}) \cdot V_\kappa(i-1,j,k-1,5) \\ & + (\frac{1}{h_x^2} + \frac{1}{h_z^2}) \cdot V_\kappa(i,j,k-1,5) + (\frac{1}{h_y^2} + \frac{1}{h_z^2}) \cdot V_\kappa(i,j,k-1,6) \\ & + (\frac{1}{h_x^2} + \frac{1}{h_z^2}) \cdot V_\kappa(i,j-1,k-1,4) + (\frac{1}{h_x^2} + \frac{1}{h_y^2}) \cdot V_\kappa(i,j-1,k-1,6) \\ & + h_x h_y h_z \cdot c_{i,j,k}, \end{aligned}$$

and now the other non-zero entries and the right-hand side.

$$
\begin{aligned}
a_{(i,j,k),(i+1,j,k)} &= -\frac{1}{h_x^2} \cdot (V_\kappa(i,j,k,1) + V_\kappa(i,j-1,k,2) + V_\kappa(i,j,k,3) \\
&\quad + V_\kappa(i,j-1,k-1,4) + V_\kappa(i,j,k-1,5) + V_\kappa(i,j-1,k-1,6)), \\
a_{(i,j,k),(i,j+1,k)} &= -\frac{1}{h_y^2} \cdot (V_\kappa(i-1,j,k,1) + V_\kappa(i,j,k,2) + V_\kappa(i-1,j,k-1,3) \\
&\quad + V_\kappa(i,j,k,4) + V_\kappa(i-1,j,k-1,5) + V_\kappa(i,j,k-1,6)), \\
a_{(i,j,k),(i,j,k+1)} &= -\frac{1}{h_z^2} \cdot (V_\kappa(i-1,j-1,k,1) + V_\kappa(i-1,j-1,k,2) + V_\kappa(i-1,j,k,3) \\
&\quad + V_\kappa(i,j,k,5) + V_\kappa(i,j,k,6) + V_\kappa(i,j-1,k,4)), \\
b_{i,j,k} &= h_x h_y h_z \cdot f_{i,j,k}.
\end{aligned}
$$

In every row, there are only seven non-zero entries of A, but again, the bandwidth is much larger. The same considerations concerning the inversion of A as in the two-dimensional case apply.

Bibliography

[1] Dietrich Braess. *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie.* Springer, Berlin, 4., ueberarb. und erw. edition, 2007.

[2] Frederico F. Campos and Nick R.C. Birkett. An efficient solver for multi-right-hand-side linear systems based on the cccg(η) method with applications to implicit time-dependent partial differential equations. *SIAM Journal of Scientific Computing*, 19(1):126–138, January 1998.

[3] Peter Deuflhard and Andreas Hohmann. *Numerische Mathematik I. Eine algorithmische Einführung.* Walter de Gruyter, Berlin, New York, 2nd edition, 1993.

[4] Lawrence C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics.* American Mathematical Society, Providence, Rhode Island, 1998.

[5] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 2.1.* High-Performance Computing Center Stuttgart, University of Stuttgart, Nobelstr.19, D-70550 Stuttgart, June 2008.

[6] Andreas Frommer. *Algorithmen und Datenstrukturen II. Parallele Algorithmen.* Bergische Universität Wuppertal, 2004/05.

[7] Martin Hanke-Burgeois. *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens.* Teubner, Wiesbaden, 2., ueberarb. und erw. edition, 2006.

[8] Mark T. Jones and Paul E. Plassmann. An improved imcomplete cholesky factorization. *ACM Transactions on Mathematical Software*, 21(1):5–17, March 1995.

[9] Friedemann Kemm. *Divergenzkorrekturen und asymptotische Untersuchungen bei der numerischen Simulation idealer magnetohydrodynamischer Strömungen.* PhD thesis, Universität Stuttgart, Stuttgart, Germany, 2006.

[10] Nipun Kwatra, Jonathan Su, Jón T. Grétarsson, and Ronald Fedkiw. A method for avoiding the acoustic time step restriction in compressible flow. *Journal of Computational Physics*, 228(11):4146–4161, June 2009.

[11] Bernhard Löw-Baselli. *A Numerical Simulation of Solar Magnetoconvection.* PhD thesis, Universität Wien, Vienna, Austria, 2008.

[12] J.A. Meijerink and H.A. van der Vorst. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *Journal of Computational Physics*, 44(1):134–155, November 1981.

[13] Andreas Meister. *Numerik linearer Gleichungssysteme.* vieweg, Wiesbaden, 3. ueberarb. edition, 2008.

[14] H. Muthsam, F. Kupka, B. Löw-Baselli, M. Langer, and P. Lenz. Antares – a numerical tool for astrophysical research – with applications to solar granulation. *NewAstronomy*, 15:460–475, 2010.

[15] Christof Obertscheider. *Modelling of solar granulation – Implementation and comparison of numerical schemes.* PhD thesis, Universität Wien, Vienna, Austria, April 2007.

[16] Rolf Rabenseifner. Parallel programming workshop. Course Material for HLRS Course 2009-E at LRZ, Garching at Munich, Universität Stuttgart, Lehrstuhl für Höchstleistungsrechnen, September 2009.

[17] Yousef Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2nd p. cm. edition, 2003.

Curriculum Vitae

Name:	Hannes Grimm-Strele
Birth:	8th of January, 1985, in Karlsruhe, Germany
Citizenship:	Austria and Germany
High school diploma:	Bismarck-Gymnasium Karlsruhe, 76133 Karlsruhe, Bismarckstr. 8-10, Germany 24th of June, 2004
Diploma studies in Mathematics:	fall term 2004 – fall term 2009 no. of terms: 11
Visting researcher:	September 2009, Max-Planck-Institut für Astrophysik, Garching bei München
Tutor for Mathematics:	2008 – 2009, University of Vienna