Universität Trier

Multigrid Optimization Methods for High Performance Computing

DISSERTATION

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt dem Fachbereich IV der Universität Trier
von Diplom-Wirtschaftsmathematiker

Christian Wagner

Trier, im August 2012

Die in dieser Dissertation verwendeten geschützten Warenzeichen sind nicht als solche gekennzeichnet. Aus dem Fehlen einer Kennzeichnung kann folglich nicht geschlossen werden, dass der entsprechende Produktname frei von Rechten Dritter ist.

Gutachter:

Prof. Dr. Volker Schulz, Universität Trier
Prof. Dr. Alfio Borzì, Universität Würzburg

Zusammenfassung

Optimale Steuerungsprobleme treten in einer Vielzahl von praktischen Anwendungen auf, zum Beispiel in der Robotersteuerung, Steuerung biologischer Prozesse, Simulation von Testfahrten und in der Form- und Topologieoptimierung. Charakterisiert werden solche Probleme durch ein Zielfunktional, welches unter bestimmten Nebenbedingungen, die durch gewöhnliche oder partielle Differentialgleichungen sowie eventuell vorhandener weiterer Beschränkungen gegeben sind, minimiert wird.
In der vorliegende Dissertation wird ein akademisches Modelproblem mit Tracking Zielfunktional und einem Regularisierungsterm sowie einer linearen, gleichmäßig elliptischen partiellen Differentialgleichung zweiter Ordnung als Nebenbedingung betrachtet. Hierfür kann Existenz und Eindeutigkeit einer optimalen Lösung gezeigt werden. Dem Paradigma 'First optimize, then discretize' folgend werden zuerst die notwendigen und hinreichenden Optimalitätsbedingung mittels Berechnung der Adjungierten hergeleitet und in Form eines Optimalitätssystems beschrieben. In einem zweiten Schritt erfolgt die Approximation der auftretenden Differentialoperatoren durch Finite Differenzen. Für die numerische Lösung des resultierenden linearen Systems wird einerseits die Implementierbarkeit von bekannten und hinreichend untersuchten Optimierungsalgorithmen, im Speziellen eines kollektiven Mehrgitterverfahrens (CSMG), auf neue Rechnerarchitekturen untersucht, andererseits werden neue Algorithmen zur Lösung von optimalen Steuerungsprobleme konstruiert, die die effiziente Nutzung mehrerer paralleler Prozessoren erlauben.
Die Implementierung dieser Algorithmen erfolgt auf einer handelsüblichen Grafikkarte (GPU): konzipiert für Grafikberechnungen, welche sich durch ein hohes Maß an Parallelisierung auszeichnen besitzen sie eine Vielzahl von Kernen und erreichen eine höhere Rechenleistung im Vergleich zu einer CPU. Die GPUs werden hier als Prototyp für zukünftige Multicore-Architekturen mit mehreren hundert Kernen betrachtet. Es zeigt sich, dass für moderate Problemgrößen eine Verbesserung in der Rechenzeit im Vergleich zu einer klassischen CPU erreichbar ist. Für größere Probleme ist der beschränkte Speicherplatz des Hauptspeichers der Grafikkarte der limitierende Faktor. Hierfür wird eine nichtüberlappende Gebietszerlegungsstrategie konstruiert, wobei mehrere GPUs bzw. CPUs parallel genutzt werden können. Diese Strategie basiert auf Vorarbeiten für elliptische Probleme und wird für optimale Steuerungsprobleme weiterentwickelt. Für eine Zerlegung in zwei Teilgebiete wird mittels einer Schur Komplement Methode das Gleichungssystem für den inneren Rand durch eine diskrete Approximation des Steklov-Poincaré Operators hergeleitet, welches dann direkt gelöst werden kann. Aufbauend auf dieser Zerlegung werden in dieser Dissertation zwei verschiedene Algorithmen für die Gebietszerlegung in mehrere Gebiete betrachtet: auf der einen Seite ein rekursiver Ansatz, auf der anderen Seite ein simultaner Ansatz. Numerische Tests vergleichen die Performance des kollektiven Mehrgitterverfahrens auf der GPU mit der CPU für verschiedene Varianten.

Acknowledgments

First of all, my greatest appreciation goes to my advisor Prof. Dr. Volker H. Schulz for his extensive support during the last three years. He gave me the opportunity to commit my research on this very interesting topic almost independently. He always took his time for inspiring discussion and for giving constructive remarks. Moreover, he gave me the opportunity to present my work at different international conferences.

My gratitute also goes to Prof. Dr. Alfio Borzì for the attendance of being the second referee within the disputation, including the duty for travelling to Trier and to Prof. Dr. Ekkehard W. Sachs, who acted as the head of the examination board.

Next, I would like to thank all the teachers and professors that contributed to my mathematical education. Here Prof. Dr. Rainer Tichatschke is highlighted, who encouraged me for an academical carrier and Rainer Hintz, who raised my interest in mathematical problems.

I would like to thank the first floor of the e-Building, my group, namely Benjamin Rosenbaum, Dr. Claudia Schillings, Martin Siebenborn, Dr. Stephan Schmidt and Roland Stoffel for creating a comfortable working atmosphere and the weekly 'Jour fixe'.

Ulf Friedrich, Dr. Christina Jager, Dr. Nils Langenberg, Andreas Madert, Matthias Schu, Martin Siebenborn, Roland Stoffel and Matthias Wagner I would like to thank for fruitful discussions, reading parts of this thesis and giving me constructive remarks.

Furthermore, I would like to thank my family, especially my parents Raimund and Christine Wagner for the - material and immaterial - support during the entire time of my academic studies, which had made many things easier.

Last but not least, I thank Daniela Nierobisch for many kinds of things - in the first instance for having so much patience and understanding for my mathematical and, above all, non-mathematical problems.

This work has been supported by the German Ministry for Education and Research (BMBF) program 'ASIL: Advanced Solvers Integrated Library' as the project 'Optimization Methods for Scalable Parallel Computers'.

Contents

1 Introduction

1.1 Motivation

Optimal control problems are optimization problems governed by ordinary or partial differential equations (PDEs). A general formulation is given by

$$\min_{(y,u)} J(y,u)$$

with subject to

$$e(y,u) = 0,$$

assuming that e_y^{-1} exists and consists of the three main elements:

- The cost functional J that models the purpose of the control on the system.
- The definition of a control function u that represents the influence of the environment of the systems.
- The set of differential equations $e(y,u)$ modeling the controlled system, represented by the state function $y := y(u)$ which depends on u.

These kind of problems are well investigated and arise in many fields of application, for example robot control, control of biological processes, test drive simulation and shape and topology optimization, see for example [7] and the references given therein. In this thesis, an academic model problem of the form

$$\min_{(y,u)} J(y,u) := \min_{(y,u)} \frac{1}{2}\|y - y_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|u\|^2_{L^2(\Omega)}$$

subject to

$$\begin{aligned} -\operatorname{div}(A \operatorname{grad} y) + cy &= f + u \quad \text{in } \Omega \\ y &= 0 \qquad \text{on } \partial\Omega \end{aligned}$$

and $u \in U_{ad}$ is considered. The objective is tracking type with a given target function y_d and a regularization term with parameter α. The control function u takes effect on the whole domain Ω. The underlying partial differential equation is assumed to be uniformly elliptic. This problem belongs to the class of *linear-quadratic elliptic control problems with distributed control*. The existence and uniqueness of an optimal solution for problems of this type is well-known and in a first step, following the paradigm 'first optimize, then discretize', the necessary and sufficient optimality conditions are derived by means of the adjoint equation which ends in a characterization of the optimal solution in form of an optimality system. In a second step,

the occurring differential operators are approximated by finite differences and the hence resulting discretized optimality system is solved with a *collective smoothing multigrid method (CSMG)*, proposed in [6]. In general, there are several optimization methods for solving the optimal control problem: an application of the implicit function theorem leads to so-called black-box approaches where the PDE-constrained optimization problem is transformed into an unconstrained optimization problem and the reduced gradient for these reduced functional is computed via the adjoint approach. Another possibilities are Quasi-Newton methods, which approximate the Hessian by a low-rank update based on gradient evaluations, Krylov-Newton methods or (reduced) SQP methods, see [7]. The use of multigrid methods for optimization purposes is motivated by its optimal computational complexity, i.e. the number of required computer iterations scales linearly with the number of unknowns and the rate of convergence, which is independent of the grid size. Originally multigrid methods are a class of algorithms for solving linear systems arising from the discretization of partial differential equations.
The main part of this thesis is devoted to the investigation of the implementability and the efficiency of the CSMG on commodity graphics cards. *GPUs (graphic processing units)* are designed for highly parallelizable graphics computations and possess many cores of SIMD-architecture, which are able to outperform the CPU regarding to computational power and memory bandwidth. Here they are considered as prototype for prospective multi-core computers with several hundred of cores. When using GPUs as streamprocessors, two major problems arise: data have to be transferred from the CPU main memory to the GPU main memory, which can be quite slow and the limited size of the GPU main memory. Furthermore, only when the streamprocessors are fully used to capacity, a remarkable speed-up comparing to a CPU is achieved.
Therefore, new algorithms for the solution of optimal control problems are designed in this thesis. To this end, a nonoverlapping domain decomposition method is introduced which allows the exploitation of the computational power of many GPUs resp. CPUs in parallel. This algorithm is based on preliminary work for elliptic problems and enhanced for the application to optimal control problems. For the domain decomposition into two subdomains the linear system for the unknowns on the interface is solved with a Schur complement method by using a discrete approximation of the Steklov-Poincaré operator. For the academic optimal control problem, the arising capacitance matrix can be inverted analytically. On this basis, two different algorithms for the nonoverlapping domain decomposition for the case of many subdomains are proposed in this thesis: on the one hand, a recursive approach and on the other hand a simultaneous approach.
Numerical test compare the performance of the CSMG for the one domain case and the two approaches for the multi-domain case on a GPU and CPU for different variants.

1.2 Outline

The present thesis is structured as follows:

In **Chapter 2** the theoretical basis for optimization with partial differential equations is established. Some functional analytic basics are recalled with the aim to introduce Sobolev spaces, where the existence and uniqueness of weak solutions of the underlying uniformly elliptic PDE can be proven as well as the existence and uniqueness of an optimal solution of

the academic optimal control problem by means of the adjoint approach. This chapter finishes with a characterization of this optimal solution in form of an optimality system.

Chapter 3 is dedicated to the *collective smoothing multigrid method (CSMG)*. In a first step, the optimality system for the unconstrained case is reformulated by eliminating the control u and the differential operators are approximated with a finite difference method. For the discretized optimality system, regularity and consistency to the continuous analogue is shown. In a second step, general linear geometric multigrid methods for linear problems with its two main components, in particular the smoothing iteration and the coarse grid correction scheme, are introduced. The CSMG means solving the optimality system for the state and the adjoint variables simultaneously in the multigrid process by using collective smoothers. Here the damped block-Jacobi relaxation and the red-black ordered block Gauss-Seidel are considered and investigated. The local Fourier analysis aims to obtain sharp convergence estimates by simplifying assumptions on the boundary conditions. This chapter ends with a convergence proof for the CSMG.

In **Chapter 4** a nonoverlapping domain decomposition method for the optimality system is introduced. Therefore, a discrete formulation of the Steklov-Poincaré operator for the two subdomain case is derived with a Schur Complement method, resulting in a linear system for the interface. This system can be solved exactly by setting up the inverse of the capacitance matrix analytically. For the nonoverlapping domain decomposition into many subdomains, this procedure is extended straightforward and two different algorithms are proposed and compared: a recursive approach and a simultaneous approach. Extensions to more general problems, like three-dimensional problems, and the use of the Schur complement as preconditioner for the interface of more general differential operators or on more general domain shapes finish this chapter.

Chapter 5 gives a brief presentation of the architecture and programming model of commodity *graphics cards (GPUs)*. It starts with a retrospection to the beginnings of GPU programming, where the only exercise of the GPU was rendering and ends with current GPUs that can perform general purpose computations. The special memory hierarchy and threads concept is compared to the architecture of a traditional CPU. In the end, the implementation of the CSMG is described and performance considerations are presented.

In **Chapter 6** the numerical results of the various algorithms are exposed. Firstly the CSMG for one domain in two and three dimensions is considered with the collective damped Jacobi relaxation as well as the collective red-black Gauss-Seidel smoother, different cycle types (V vs. W cycle), the influence of the total number of smoothing steps and a comparison of double and single precision performance on both architectures, GPU and CPU. The robustness with respect to the relaxation parameter α is investigated. This chapter finishes with a comparison of both algorithms for the nonoverlapping domain decomposition in many subdomains, the recursive and the simultaneous approach.

The last **Chapter 7** recapitulates the thesis with conclusions and remarks and finishes with an outlook on further investigations.

2 Optimal Control of Uniformly Elliptic Partial Differential Equations

In this first introductory chapter the model optimal control problem is presented. For this purpose, this chapter is divided into five sections: In the first Section 2.1, the analytical background, in particular some basic functional analytic terms and definitions, is recapitulated. This preliminary work is necessary to prove the existence and uniqueness of a solution of the underlying elliptic *partial differential equation (PDE)* introduced in Section 2.2. The academic tracking type optimal control problem is defined in Section 2.3, the existence and uniqueness of a solution of this problem is shown in Section 2.4 and in a final step, the unique optimal solution of this optimal control problem is characterized in form of an optimality system, derived in Section 2.5.

2.1 Basics of Functional Analysis

For proving the existence and uniqueness of the elliptic partial differential equation in suitable function spaces, some elementary functional analytic definitions and theorems are recalled in this section. The primary objective is to introduce Sobolev spaces, where the existence of weak solutions of particular PDEs can be shown. Wider and more detailed expositions can be found in any book on linear functional analysis, e.g. [2], [23] or [33].

2.1.1 Banach and Hilbert Spaces

Firstly several basics on Banach and Hilbert spaces are recalled. The term of a *linear space* (also: *vector space*) is considered as well-known and not defined in this thesis.

DEFINITION 2.1 (Norm, normed linear space)**.**
Let U be a real linear space. A mapping $\|\cdot\| : U \to [0, \infty)$ with the properties

(N1) $\|u\| \geq 0 \quad \forall u \in U, \quad \|u\| = 0 \Leftrightarrow u = 0$

(N2) $\|\lambda u\| = |\lambda| \|u\| \quad \forall \lambda \in \mathbb{R}, u \in U$

(N3) $\|u + v\| \leq \|u\| + \|v\| \quad \forall u, v \in U$

is called a *norm* on U. A linear space with norm is called *normed linear space* $(U, \|\cdot\|)$. The definition of $|\cdot|$ will be given in Definition 2.4.

DEFINITION 2.2 (Completeness, Banach space)**.**

1. A normed linear space $(U, \|\cdot\|)$ is *complete*, if any Cauchy sequence $\{u_n\}_{n\in\mathbb{N}} \subset U$ has a limit $u \in U$, i.e. if

$$\lim_{m,n\to\infty} \|u_m - u_n\| = 0$$

then there is some $u \in U$ with

$$\lim_{n\to\infty} \|u_n - u\| = 0.$$

2. A normed real linear space $(U, \|\cdot\|)$ is called *real Banach space* if it is complete.

DEFINITION 2.3 (Scalar product, Pre-Hilbert space, Hilbert space)**.**
Let H be a real linear space.

1. A mapping $(\cdot,\cdot) : H \times H \to \mathbb{R}$ with the properties

(H1) $(u,u) \geq 0 \quad \forall u \in H, \quad (u,u) = 0 \Leftrightarrow u = 0$

(H2) $(u,v) = (v,u) \quad \forall u, v \in H$

(H3) $(\alpha u + \beta v, w) = \alpha(u,w) + \beta(v,w) \quad \forall \alpha, \beta \in \mathbb{R}, u, v, w \in H$

is called *scalar product* or *inner product*.

2. A real vector space H with the scalar product $(\cdot,\cdot)$ is called *Pre-Hilbert space* $(H, (\cdot,\cdot))$.
3. A Pre-Hilbert space $(H, (\cdot,\cdot))$ is called *Hilbert space* if it is complete with its associated norm $\|u\| := \sqrt{(u,u)}$, $u \in H$.

Every Pre-Hilbert space is a normed linear space by setting

$$\|u\| := \sqrt{(u,u)}, \quad u \in H.$$

An important example for a finite-dimensional Hilbert space is the $\mathbb{R}^d$ with the euclidean scalar product $(\cdot,\cdot)_2$.

DEFINITION 2.4 (Euclidean norm, euclidean scalar product, absolut value)**.**
Let $x, y \in \mathbb{R}^d$.

1. The *euclidean scalar product* $(\cdot,\cdot)_2$ of x and y is given by

$$(x,y)_2 = \sum_{i=1}^{d} x_i y_i.$$

2. The *euclidean norm* $\|\cdot\|_2$ of x is given by

$$\|x\|_2 := \sqrt{(x,x)_2} = \left(\sum_{i=1}^{d} x_i^2\right)^{1/2}.$$

3. For $d = 1$ the *absolut value* of x is given by

$$|x| := \|x\|_2.$$

In Pre-Hilbert spaces, the Cauchy-Schwarz inequality holds:

LEMMA 2.5 (Cauchy-Schwarz inequality)**.**
Let $(H, (\cdot,\cdot))$ be a Pre-Hilbert space. Then the *Cauchy-Schwarz inequality*

$$|(u,v)| \leq \|u\|\|v\| \quad \forall u, v \in H$$

holds.

Proof. [23, Lemma 2.21]. □

2.1.2 Linear Mappings

Linear partial differential operators define linear mappings between function spaces. In this section, some of its properties are presented.

DEFINITION 2.6 (Linear operator)**.**
Let $(U, \|\cdot\|_U), (V, \|\cdot\|_V)$ be normed real linear spaces.

1. A mapping

$$\begin{aligned} F : U &\to V, \\ u &\mapsto F(u) \end{aligned}$$

is called

- *continuous at* $u \in U$, if for every sequence $\{u_n\}_{n\in\mathbb{N}} \subset U$ converging to $u \in U$ it holds that
$$\lim_{n\to\infty} \|u_n - u\|_U = 0 \Rightarrow \lim_{n\to\infty} \|F(u_n) - F(u)\|_V = 0.$$
- *continuous*, if F is continuous at every $u \in U$.
- *linear*, if
$$F(\lambda u + \mu v) = \lambda F(u) + \mu F(v) \quad \forall u, v \in U, \lambda, \mu \in \mathbb{R}.$$
- *bounded*, if F is linear and there exists a constant $C_F \geq 0$ such that
$$\|F(u)\|_V \leq C_F \|u\|_U \quad \forall u \in U.$$

2. $L(U,V)$ denotes the space of all linear and continuous operators $F : U \to V$.
3. For $F : U \to V$ linear and continuous define the operator norm by
$$\|F\|_{L(U,V)} := \sup_{\|u\|_U = 1} \|F(u)\|_V.$$

LEMMA 2.7.
Let $(U, \|\cdot\|_U), (V, \|\cdot\|_V)$ be normed real linear spaces. A linear operator $F : U \to V$ is bounded if and only if F is continuous.

Proof. [23, Lemma 2.8]. □

THEOREM 2.8.
Let $(U, \|\cdot\|_U), (V, \|\cdot\|_V)$ be normed real linear spaces. Then $(L(U,V), \|\cdot\|_{L(U,V)})$ is a normed space. If $(V, \|\cdot\|_V)$ is a Banach space, then $(L(U,V), \|\cdot\|_{L(U,V)})$ is a Banach space.

Proof. [23, Theorem 2.10]. □

A useful result for the construction of continuous linear mappings is

THEOREM 2.9.
Let $(U, \|\cdot\|_U), (V, \|\cdot\|_V)$ be Banach spaces, $M \subset U$ dense and $T : M \to V$ continuous and linear. Then there exists a unique $\tilde{T} \in L(U,V)$ with $\tilde{T}|_M = T$ and $\|T\|_{L(M,V)} = \|\tilde{T}\|_{L(U,V)}$.

Proof. [23, Theorem 2.14]. □

Hence any operator is determined uniquely by its action on a dense subspace.

DEFINITION 2.10 (Linear functional, dual space)**.**
Let $(U, \|\cdot\|_U)$ be a normed real linear space, $u \in U$.

1. A linear mapping $u^* : U \to \mathbb{R}, u \mapsto u^*(u)$ is called *linear functional*.
2. The space $U^* := L(U, \mathbb{R})$ is called *dual space* of U and is (by Theorem 2.8) a Banach space with the operator norm
$$\|u^*\|_{U^*} := \sup_{\|u\|_U = 1} |u^*(u)|.$$
3. $\langle \cdot, \cdot \rangle_{U^*,U} := u^*(u)$ is called *dual pairing* of U^* and U.

The *Riesz representation theorem* gives an important statement about the concrete representation of linear and continuous functionals in Hilbert spaces and the structure of the dual space.

THEOREM 2.11 (Riesz representation theorem)**.**
Let $(U, (\cdot,\cdot)_U)$ be a real Hilbert space and $u^* : U \to \mathbb{R}$ a linear functional. Then there exists a unique $u \in U$ with
$$(v, u)_U = \langle u^*, v \rangle_{U^*,U} \quad \forall v \in U.$$
Furthermore $\|u^*\|_{U^*} = \|u\|_U$.

Proof. [23, Theorem 2.25]. □

2.1.3 Lebesgue Spaces

The next topic are the Sobolev spaces $W^{k,p}(\Omega)$. As the classical function spaces $C^k(\overline{\Omega})$ are not complete with respect to the $\|\cdot\|_2$-norm, the concept of weak partial derivatives is introduced and these classical spaces are extended to the Sobolev spaces $W^{k,p}(\Omega)$, where the existence of weak solutions of certain partial differential equations can be proven. For this purpose, the Lebesgue spaces $L^p(\Omega)$ are defined in this subsection.

DEFINITION 2.12 (Area)**.**

1. $\Omega \subset \mathbb{R}^d$ is called *area*, if Ω is open and connected.
2. $\partial\Omega$ is the *boundary of* Ω.
3. $\overline{\Omega} := \Omega \cap \partial\Omega$ denotes the *closure of* Ω.

DEFINITION 2.13 (Multi-index, support, function spaces)**.**

1. For $\Omega \subset \mathbb{R}^d$, define
$$C(\Omega) := \{u : \Omega \to \mathbb{R} : u \text{ continuous}\}.$$
2. Let $\Omega \subset \mathbb{R}^d$ be open, $u : \Omega \to \mathbb{R}$. For a multi-index $\alpha = (\alpha_1, ..., \alpha_n) \in \mathbb{N}_0^n$ its order is defined by $|\alpha| := \sum_{i=1}^n \alpha_i$ and the $|\alpha|$-th order partial derivative at $x \in \Omega$ is given by
$$D^\alpha u(x) := \frac{\partial^{|\alpha|} u}{\partial x_1^{\alpha_1} \cdots \partial x_n^{\alpha_n}}(x).$$
3. Define
$$C^k(\Omega) := \{u \in C(\Omega) : D^\alpha u \in C(\Omega) \text{ for } |\alpha| \leq k\}.$$
4. For $\Omega \subset \mathbb{R}^d$ open and bounded define
$$C^k(\overline{\Omega}) := \{u \in C^k(\Omega) : D^\alpha u \text{ has a continuous extension to } \overline{\Omega} \text{ for } |\alpha| \leq k\}.$$
5. For $\Omega \subset \mathbb{R}^d$, $u : \Omega \to \mathbb{R}$, the set
$$\operatorname{supp} u = \overline{\{x \in \Omega : u(x) \neq 0\}}$$
is called *support of* u.
6. Define
$$C^\infty(\Omega) = \bigcap_{m=0}^{\infty} C^m(\Omega) \text{ and}$$
$$C_0^\infty(\Omega) = \{u \in C^\infty(\Omega) : \operatorname{supp}(u) \subset \Omega \text{ compact}\}$$

LEMMA 2.14.
Let $\Omega \subset \mathbb{R}^d$, $u : \Omega \to \mathbb{R}$.

1. If Ω is bounded then $C(\overline{\Omega})$ is a Banach space with the *supremum*-norm
$$\|u\|_{C(\overline{\Omega})} := \sup_{u \in \overline{\Omega}} |u(x)|.$$
2. If Ω is open and bounded, the spaces $C^k(\overline{\Omega})$ are Banach spaces with the norm
$$\|u\|_{C^k(\overline{\Omega})} := \sum_{|\alpha| \leq k} \|D^\alpha u\|_{C(\overline{\Omega})}.$$

Proof. [2, Theorem/Definition 1.2] and [2, Theorem/Definition 1.5]. □

The theory for the solvability of partial differential equations needs some assumptions regarding the area Ω. The boundary $\partial\Omega$ has to be sufficiently smooth.

DEFINITION 2.15 (Lipschitz continuity, Lipschitz boundary, normal derivative)**.**

1. Let $\Omega \subset \mathbb{R}^d$ be open. A function $F : \bar{\Omega} \to \mathbb{R}$ is *Lipschitz continuous* if there is a constant $C_F > 0$ such that
$$|F(x) - F(y)| \leq C_F \|x - y\| \quad \forall x, y \in \bar{\Omega}.$$

2. Let $\Omega \subset \mathbb{R}^d$ be open. For $k \in \mathbb{N}_0$ let
$$C^{k,1}(\overline{\Omega}) = \{u \in C^k(\overline{\Omega}) : D^\alpha u \text{ Lipschitz continuous for } |\alpha| = k\}.$$

3. Let $\Omega \subset \mathbb{R}^d$ be open and bounded. Ω has a $C^{k,1}$-boundary, $k \in \mathbb{N}_0 \cup \{\infty\}$, if for any $x \in \partial\Omega$ there exists $r > 0$, $j \in \{1, \ldots, n\}$, $\sigma \in \{-1, +1\}$, and a function $\gamma \in C^{k,1}(\mathbb{R}^{n-1})$ such that
$$\Omega \cap B(x; r) = \{y \in B(x; r) : \sigma\gamma_j < \gamma(y_1, \ldots y_{j-1}, y_{j+1}, \ldots, y_n)\},$$
where
$$B(x; r) := \{y \in \mathbb{R}^d : \|y - x\| < r\}$$
denotes the open ball around x with radius r.
A $C^{0,1}$-boundary is called *Lipschitz boundary*. For short, say $\partial\Omega$ is $C^{k,1}$.

4. If $\partial\Omega$ is $C^{0,1}$ then a unit outer normal field $\nu : \partial\Omega \to \mathbb{R}^n$ can be defined almost everywhere, where $\nu(x)$ with $\|\nu(x)\| = 1$ is the outward pointing *unit normal* of $\partial\Omega$ at x.

5. Let $\partial\Omega$ be $C^{0,1}$, $u : \overline{\Omega} \to \mathbb{R}$. The directional derivative
$$\frac{\partial u}{\partial \nu} := \nu(x) \cdot \nabla u(x), \quad x \in \partial\Omega$$
is called the *normal derivative of* u. ∇ denotes the gradient, later defined in Definition 2.28.

Now some standard spaces of integrable functions based on the Lebesgue integral are introduced. For the definition of the Lebesgue integral see e.g. [23, Chapter 4.1].

DEFINITION 2.16 (L^p spaces)**.**
Let $\Omega \subset \mathbb{R}^d$ be bounded.

1. For $1 \leq p < \infty$, define the space
$$\mathcal{L}^p(\Omega) := \left\{u : \Omega \to \mathbb{R} : \int_\Omega |u(x)|^p \, \mathrm{dx} < \infty\right\}$$
with the semi-norm
$$\|u\|_{\mathcal{L}^p(\Omega)} := \left(\int_\Omega |u(x)|^p \, \mathrm{dx}\right)^{1/p}.$$

2. Define
$$\operatorname*{ess\,sup}_{x\in\Omega} |u(x)| := \inf\{\alpha \geq 0 : \lambda(\{|u| > \alpha\}) = 0\}$$
as the *essential supremum* of a function u. λ denotes the Lebesgue measure.

3. For $p = \infty$ define

$$\mathcal{L}^\infty(\Omega) := \left\{ u : \Omega \to \mathbb{R} : \operatorname*{ess\,sup}_{x\in\Omega} |u(x)| < \infty \right\}$$

with the semi-norm

$$\|u\|_{\mathcal{L}^\infty(\Omega)} := \operatorname*{ess\,sup}_{x\in\Omega} |u(x)|.$$

Since there are measurable functions $u : \Omega \to \mathbb{R}$, $u \neq 0$, with $\|u\|_{\mathcal{L}^p(\Omega)} = 0$, the equivalence relation

$$u \sim v \text{ in } \mathcal{L}^p(\Omega) :\Leftrightarrow \|u - v\|_{\mathcal{L}^p(\Omega)} = 0 \Leftrightarrow u = v \text{ almost everywhere}$$

is used to define $L^p(\Omega) := \mathcal{L}^p(\Omega)/\sim$ as the space of equivalence classes of almost everywhere identical functions, equipped with the norm $\|\cdot\|_{L^p(\Omega)} := \|\cdot\|_{\mathcal{L}^p(\Omega)}$ for $1 \leq p \leq \infty$. Therefore the norm is well-defined for $p \in [1, \infty]$. For more details see [34, Definition 1.11/Lemma 1.2] or [2, Theorem/Definition 1.13].

THEOREM 2.17.
The spaces $L^p(\Omega)$, $1 \leq p \leq \infty$ are Banach spaces and for $p = 2$, the space $L^2(\Omega)$ equipped with the scalar product

$$(u, v)_{L^2(\Omega)} := \int_\Omega u(x)\overline{v(x)}\,\mathrm{dx}$$

for $u, v \in L^p(\Omega)$ is a Hilbert space.

Proof. [23, Theorem 4.17/Corollar 4.18] □

As assumed that $v(x) \in \mathbb{R}$ for all x, the complex conjugate $\overline{v(x)}$ of $v(x)$ is $v(x)$ itself.

2.1.4 Weak Derivatives

Using the integration by parts formula, the concept of the classical derivative can be generalized by introducing weak derivatives. Recall Green's formula

THEOREM 2.18 (Green's formula)**.**
Let $\Omega \subset \mathbb{R}^d$ be open and bounded with $C^{0,1}$-boundary. Then for all $u, v \in C^1(\overline{\Omega})$

$$\int_\Omega D_i u(x)v(x)\,\mathrm{dx} = -\int_\Omega u(x)D_i v(x)\,\mathrm{dx} + \int_{\partial\Omega} u(x)v(x)\nu_i(x)\,\mathrm{dS},$$

where $\nu_i(x)$ is the outward pointing unit normal in $x \in \partial\Omega$, D_i the partial derivative in x_i-direction and dS the Lebesgue surface measure on $\partial\Omega$.

Proof. [34, Theorem 1.9] □

For $v \in C^k(\overline{\Omega})$ and $u \in C_0^k(\Omega)$ as well as for a multi-index α with $|\alpha| \leq k$ with repeated integration by parts one obtains

$$\int_\Omega D^\alpha u(x)v(x)\,\mathrm{dx} = (-1)^\alpha \int_\Omega u(x)D^\alpha v(x)\,\mathrm{dx}.$$

This leads to the idea of the weak derivative. For the purpose of generalization, the set of every in Ω local integrable functions $L^1_{loc}(\Omega)$ is defined.

DEFINITION 2.19 (Local integrable function)**.**
Let $\Omega \subset \mathbb{R}^d$ be open. Define

$$\mathcal{L}^p_{loc}(\Omega) := \{u : \Omega \to \mathbb{R} \text{ Lebesgue-measurable: } u \in \mathcal{L}^p(K) \text{ for all } K \subset \Omega \text{ compact}\},$$

and, as remarked before, set again $L^p_{loc}(\Omega) := \mathcal{L}^p_{loc}(\Omega)/\sim$.

DEFINITION 2.20 (Weak derivative)**.**
Let $\Omega \subset \mathbb{R}^d$ be open, $u \in L^1_{loc}(\Omega)$. If there exists a function $v \in L^1_{loc}(\Omega)$ such that

$$\int_\Omega v(x)\varphi(x)\,\mathrm{dx} = (-1)^{|\alpha|}\int_\Omega u(x)D^\alpha\varphi(x)\,\mathrm{dx} \quad \forall \varphi \in C_0^\infty(\Omega),$$

then $D^\alpha u := v$ is called *α-th weak partial derivative of u*.

As the classical derivative also satisfies this equation, the concept of weak derivatives is consistent.

LEMMA 2.21.
If existent, the weak derivative is determined uniquely. If a function is differentiable in the classical way, it is also weakly differentiable and the derivatives coincide.

Proof. [23, Lemma 5.4]. □

2.1.5 Sobolev Spaces

Basing on weak derivatives $D^\alpha u \in L^p(\Omega)$, the Sobolev spaces $W^{k,p}(\Omega)$ are defined. These are subspaces of the Lebesgue spaces $L^p(\Omega)$, consisting of a linear space of functions $u \in L^p(\Omega)$.

DEFINITION 2.22 (Sobolev spaces)**.**
Let $\Omega \subset \mathbb{R}^d$ be open. For $k \in \mathbb{N}_0$, $p \in [1, \infty]$ the spaces

$$W^{k,p}(\Omega) = \{u \in L^p(\Omega) : u \text{ has weak derivatives } D^\alpha u \in L^p(\Omega) \text{ for all } |\alpha| \le k\}$$

are called Sobolev spaces $W^{k,p}(\Omega)$ and are equipped with the norms

$$\|u\|_{W^{k,p}(\Omega)} := \left(\sum_{|\alpha|\le k} \|D^\alpha u\|^p_{L^p(\Omega)}\right)^{1/p}, \qquad p \in [1,\infty)$$

$$\text{and } \|u\|_{W^{k,\infty}(\Omega)} := \sum_{|\alpha|\le k} \|D^\alpha u\|_{L^\infty(\Omega)}, \qquad p = \infty.$$

As the case $p = 2$ is of particular interest, define $H^k(\Omega) := W^{k,2}(\Omega)$. For $k = 0$, define $W^{0,p}(\Omega) := L^p(\Omega)$ and $H^0(\Omega) := L^2(\Omega)$.

THEOREM 2.23.
Let $\Omega \subset \mathbb{R}^d$ be open, $k \in \mathbb{N}_0$, and $p \in [1, \infty]$. Then $W^{k,p}(\Omega)$ is a Banach space. Moreover, the space $H^k(\Omega) = W^{k,2}(\Omega)$ is a Hilbert space with inner product

$$(u, v)_{H^k(\Omega)} = \sum_{|\alpha| \leq k} (D^\alpha u, D^\alpha v)_{L^2(\Omega)}.$$

Proof. [23, Theorem 5.10/Corollar 5.11]. □

The next theorem is the main result of this section. As for $1 \leq p < \infty$ functions in $W^{k,p}(\Omega)$ can be approximated by functions in $C^\infty(\Omega) \cap W^{k,p}(\Omega)$, most of the properties of classically differentiable functions are maintained for Sobolev functions.

THEOREM 2.24 (Meyers and Serrin)**.**
Let $\Omega \subset \mathbb{R}^d$ be open. The set $C^\infty(\Omega) \cap W^{k,p}(\Omega)$, $k \in \mathbb{N}_0, 1 \leq p < \infty$, is dense in $W^{k,p}(\Omega)$. Hence, $W^{k,p}(\Omega)$ is the completion of $\{u \in C^\infty(\Omega) : \|u\|_{W^{k,p}(\Omega)} < \infty\}$ with respect to the norm $\|\cdot\|_{W^{k,p}(\Omega)}$.

Proof. [23, Theorem 5.16] □

If Ω has Lipschitz boundaries, a stronger proposition can be stated.

THEOREM 2.25.
Let Ω be bounded with $C^{0,1}$-boundary. Then $C^\infty(\overline{\Omega})$ is dense in $W^{k,p}(\Omega)$, $k \in \mathbb{N}_0, 1 \leq p < \infty$.

Proof. [23, Theorem 6.7]. □

To take account of the homogeneous boundary conditions, define

DEFINITION 2.26.
Let $\Omega \subset \mathbb{R}^d$ be open. For $k \in \mathbb{N}_0, p \in [1, \infty]$ denote by $W_0^{k,p}(\Omega)$ the closure of $C_0^\infty(\Omega)$ in $W^{k,p}(\Omega)$.

COROLLAR 2.27.
Let $\Omega \subset \mathbb{R}^d$ be open, $k \in \mathbb{N}_0$, and $p \in [1, \infty]$. The space $W_0^{k,p}(\Omega)$ equipped with the same norm as $W^{k,p}(\Omega)$ is a Banach space. The space $H_0^k(\Omega) = W_0^{k,2}(\Omega)$ is a Hilbert space.

2.2 Weak Solutions of Uniformly Elliptic Partial Differential Equations

In this section a short sketch of the concept of weak solutions of uniformly elliptic partial differential equations with homogeneous Dirichlet boundaries is given. For a more detailed survey, see e.g. [2],[23],[34] and [38].

2.2.1 Uniformly Elliptic Partial Differential Equations

Differential operators are operators defined as a function of the differentiation operator.

DEFINITION 2.28 (Differential operators)**.**
Let $\Omega \subset \mathbb{R}^d$ be open, $\rho : \Omega \to \mathbb{R}$ differentiable and $u : \Omega \to \mathbb{R}^d$ differentiable, $d = 2, 3$. Then define the differential operators

1. *gradient*

$$\nabla\rho := \operatorname{grad}\rho := \begin{pmatrix} \frac{\partial\rho}{\partial x_1} \\ \frac{\partial\rho}{\partial x_2} \end{pmatrix} \quad \text{resp.} \quad \begin{pmatrix} \frac{\partial\rho}{\partial x_1} \\ \frac{\partial\rho}{\partial x_2} \\ \frac{\partial\rho}{\partial x_3} \end{pmatrix}$$

2. *divergence*

$$\nabla \cdot u := \operatorname{div} u := \frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} \quad \text{resp.} \quad \frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} + \frac{\partial u_3}{\partial x_3}$$

3. *Laplace operator*

$$\Delta\rho := \nabla \cdot \nabla\rho := \operatorname{div}\operatorname{grad}\rho := \frac{\partial^2\rho}{\partial x_1^2} + \frac{\partial^2\rho}{\partial x_2^2} \quad \text{resp.} \quad \frac{\partial^2\rho}{\partial x_1^2} + \frac{\partial^2\rho}{\partial x_2^2} + \frac{\partial^2\rho}{\partial x_3^2}.$$

DEFINITION 2.29 (General second order differential operator with divergence structure)**.**
Let $\Omega \subset \mathbb{R}^d$ be open. For $u : \Omega \to \mathbb{R}$ define the general second order differential operator with divergence structure

$$\begin{aligned} Lu :&= -\sum_{i,k=1}^{d} \frac{\partial}{\partial x_i}\left(a_{ik}\frac{\partial}{\partial x_k}u\right) + cu \\ &= -\nabla \cdot (A\nabla u) + cu \end{aligned} \tag{2.1}$$

where $c \in L^\infty(\Omega)$, $c \geq 0$ and $a_{ik} \in L^\infty(\Omega)$, $a_{ik} = a_{ki}$ for $i, k = 1, \ldots d$.

In the special case $A = I_d$, I_d the $d \times d$ identity matrix and $c \equiv 0$, the operator L simplifies to the Laplace operator $-\Delta$.

DEFINITION 2.30 (Ellipticity)**.**
The operator

$$L := -\sum_{i,k=1}^{d} a_{ik}\frac{\partial}{\partial x_i}\frac{\partial}{\partial x_k} + c$$

is said to be *uniformly elliptic in* Ω if there is a constant $\alpha > 0$ such that it holds

$$\xi^T A(x)\xi \geq \alpha\|\xi\|_2^2$$

for all $\xi \in \mathbb{R}^d$, $A(x) := [a_{ik}(x)]_{ik}$, for almost all $x \in \Omega \subset \mathbb{R}^d$ and $c(x) \geq 0$ almost everywhere .

In the following the second order differential operator (2.1) is always assumed to be uniformly elliptic in Ω. Now a general boundary value problem with homogeneous Dirichlet data is introduced.

DEFINITION 2.31 (Boundary value problem with homogeneous Dirichlet data)**.**
Let $\Omega \subset \mathbb{R}^d$ be open, bounded and $f \in L^2(\Omega)$. A *general boundary value problem with homogeneous Dirichlet data* is defined by

$$\begin{aligned} Lu &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{2.2}$$

DEFINITION 2.32 (Classical solution)**.**
If $u \in C^2(\Omega) \cap C^0(\partial\Omega)$ is a solution of problem (2.2), u is called *classical solution*.

2.2.2 Variational Formulation

As merely $f \in L^2(\Omega)$, it is not excluded that the right-hand side in (2.2) is discontinuous. For example, source terms can occur, where f acts only on a subset of Ω. A classical solution exists at best for a continuous right-hand side, therefore a generalization of the concept of a classical solution is needed. This is based on the variational formulation of the homogeneous Dirichlet boundary problem, where a weak solution $u \in H_0^1(\Omega)$ is wanted.

DEFINITION 2.33 (Variational formulation, weak solution)**.**
A function $u \in H_0^1(\Omega)$ is called *weak solution* of the boundary value problem (2.2) if it satisfies the *variational formulation* (or also *weak formulation*)

$$\int_\Omega \left(\sum_{i,k=1}^{d} a_{ik} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_k} + cuv \right) \mathrm{dx} = \int_\Omega fv \,\mathrm{dx} \quad \forall v \in H_0^1(\Omega). \tag{2.3}$$

The weak formulation can be obtained by multiplying both sides with a test function $v \in H_0^1(\Omega)$, integrating over Ω and applying the integration by parts formula. If a classical solution exists, it certainly satisfies the weak formulation. For a more general treatment in a wider framework, the following abstract notation is introduced.

NOTATION 2.34.
Let

$$\begin{aligned} V &:= H_0^1(\Omega), \\ a(u,v) &:= \int_\Omega \left(\sum_{i,k=1}^{d} a_{ik} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_k} + cuv \right) \mathrm{dx}, \quad u, v \in V, \\ F(v) &:= \int_\Omega fv \,\mathrm{dx}, \quad v \in V. \end{aligned}$$

Here $a : V \times V \to \mathbb{R}$ is a bilinear form, $F \in V^*$ is a linear functional on V and a solution of the weak formulation (2.3) can be written as

$$\text{Find } u \in V : \quad a(u,v) = F(v) \quad \forall v \in V.$$

As the bilinear form $a(\cdot,\cdot)$ is symmetric by the assumptions in Definition 2.29, the Lax-Milgram Lemma is an immediate consequence of the Riesz representation theorem 2.11. It is the main tool to prove existence and uniqueness of a solution of the variational formulation.

LEMMA 2.35 (Lax-Milgram)**.**
Let $(V, (\cdot,\cdot)_V)$ be a real Hilbert space and let $a : V \times V \mapsto \mathbb{R}$ be a bilinear form which has the two properties:
There exists constants α_0 and $\beta_0 > 0$ with

1. $|a(u,v)| \leq \alpha_0 \|u\|_V \|v\|_V \quad \forall u, v \in V$ boundedness
2. $a(u,u) \geq \beta_0 \|u\|_V^2 \quad \forall u \in V$ V-coercivity.

Then for any bounded linear functional $F \in V^*$ the variational inequality

$$a(u,v) = F(v) \quad \forall v \in V$$

has a unique solution $u \in V$. Moreover, u satisfies

$$\|u\|_V \leq \frac{1}{\beta_0} \|F\|_{V^*}.$$

Proof. [2, Theorem 4.2]. □

To verify the requirements for the Lax-Milgram Lemma, the Poincaré-Friedrichs inequality is helpful. For this proposition, the space $H_0^1(\Omega)$ is essential as the proposition does not hold in general spaces, e.g. $H^1(\Omega)$.

THEOREM 2.36 (Poincaré-Friedrichs inequality)**.**
Let $\Omega \subset \mathbb{R}^d$ be open and bounded. Then there is $C_\Omega > 0$, only depending on Ω, such that

$$\int_\Omega u^2 \, \mathrm{dx} \leq C_\Omega \int_\Omega |\nabla u|^2 \, \mathrm{dx} \quad \forall u \in H_0^1(\Omega).$$

Proof. [2, Theorem 4.7]. □

The following lemma is a preparation for the proof of existence and uniqueness of a solution of the boundary value problem.

LEMMA 2.37.

1. The mapping

$$a : H_0^1(\Omega) \times H_0^1(\Omega) \to \mathbb{R}$$
$$(y,v) \mapsto \int_\Omega \left(\sum_{i,k=1}^{d} a_{ik} \frac{\partial y}{\partial x_i} \frac{\partial v}{\partial x_k} + cyv \right) \mathrm{dx}$$

   is bilinear and bounded:
$$|a(y,v)| \leq \|y\|_{H^1(\Omega)} \|v\|_{H^1(\Omega)}.$$

2. For $f \in L^2(\Omega)$, the mapping

$$m : H_0^1(\Omega) \to \mathbb{R}$$
$$v \mapsto \int_\Omega fv \, \mathrm{dx}$$

is linear and bounded:

$$\left| \int_\Omega f v \, \mathrm{dx} \right| = (f, v)_{L^2(\Omega)} \leq \|f\|_{L^2(\Omega)} \|v\|_{L^2(\Omega)} \leq \|f\|_{L^2(\Omega)} \|v\|_{H_0^1(\Omega)}.$$

Proof. [23, p.135]. □

2.2.3 Existence and Uniqueness of a Solution for the Boundary Value Problem

These preliminary results at hand, existence and uniqueness for solutions of boundary value problems with homogeneous Dirichlet data can be proven.

THEOREM 2.38 (Existence, uniqueness and continuity for the general Dirichlet problem)**.** Let $\Omega \subset \mathbb{R}^d$ be a bounded Lipschitz area, $c \in L^\infty(\Omega)$ and L be a uniformly elliptic second order operator according to (2.1). The problem (2.2) has for all $f \in L^2(\Omega)$ a unique weak solution $u \in H_0^1(\Omega)$, characterized by (2.3). This solution satisfies

$$\|u\|_{H^1(\Omega)} \leq C_D \|f\|_{L^2(\Omega)} \tag{2.4}$$

where C_D depends only on a_{ij}, c_0, Ω.

Proof. Application of the Lax-Milgram Lemma 2.35 and Lemma 2.37. □

With some more restrictive conditions on the smoothness of the boundary and/or the given data, a higher regularity than $u \in H_0^1(\Omega)$ can be achieved, see Chapter 3.1.

2.3 Optimal Control Problem

Optimal control problems are optimization problems governed by ordinary or partial differential equations. For the formulation, the following terms are required, see [6].

- The definition of a control function u that represents the influence of the environment on the systems.
- The set of differential equations modeling the controlled system, represented by the state function $y := y(u)$, depending on u.
- The cost functional J that models the purpose of the control on the system.

In this thesis, the following academic model problem is considered:

1. The control function u is *distributed*, i.e. it takes effect on the whole domain.
2. The underlying partial differential equation is *uniformly elliptic*.
3. The objective J is *tracking type* with a *regularization term*.

This problem belongs to the class of linear-quadratic elliptic control problems with distributed control.

DEFINITION 2.39 (Optimal control problem)**.**
The optimal control problem is given by

$$\min_{(y,u)} J(y,u) := \min_{(y,u)} \frac{1}{2}\|y - y_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|u\|^2_{L^2(\Omega)}$$

subject to

$$\begin{aligned} -\operatorname{div}(A \operatorname{grad} y) + cy &= f + u \quad \text{in } \Omega \\ y &= 0 \qquad \text{on } \partial\Omega \end{aligned} \tag{2.5}$$

and

$$u_a(x) \leq u(x) \leq u_b(x) \text{ almost everywhere in } \Omega$$

and the additional assumption:
Let $\Omega \subset \mathbb{R}^d$, $d = 2, 3$ be a bounded Lipschitz area, the desired state $y_d \in L^2(\Omega)$, $\alpha \in \mathbb{R}$, $\alpha \geq 0$ regularization parameter and $u_a, u_b \in L^2(\Omega)$ with $u_a(x) \leq u_b(x)$ for almost all $x \in \Omega$, $c \geq 0$ and $c \in L^\infty(\Omega)$. $u \in L^2(\Omega)$ is the *control* and y the *state* variable.

For $A = I_d$, $I_d \in \mathbb{R}^{d\times d}$ the identity matrix and $c(x) \equiv 0$, (2.5) simplifies to the d-dimensional Poisson equation $-\Delta y = f + u$.

DEFINITION 2.40 (Set of feasible controls)**.**
The set of feasible controls is denoted by

$$U_{ad} = \left\{u \in L^2(\Omega) : u_a(x) \leq u(x) \leq u_b(x) \text{ almost everywhere in } \Omega\right\}.$$

U_{ad} is a nonempty, convex and closed subset of $L^2(\Omega)$.

As shown in Section 2.3, for every $u \in U_{ad}$ there exists a unique weak solution $y \in H^1_0(\Omega)$ for the underlying partial differential equation. An optimal control is characterized by the following condition.

DEFINITION 2.41 (Optimal control)**.**
A control $\bar{u} \in U_{ad}$ is optimal with $\bar{y} = y(\bar{u})$ appropriate optimal state, if

$$J(\bar{y}, \bar{u}) \leq J(y, u) \quad \forall u \in U_{ad}, y = y(u) \in H^1_0(\Omega).$$

2.4 Existence and Uniqueness of a Solution of the Optimal Control Problem

In this section the existence and uniqueness of a solution of the optimal control problem will be shown. For this purpose, the approach presented in [52] is followed. Another way to achieve this aim in a more general functional setting is described [7] and [34]. Firstly, the problem is reformulated into an on u reduced optimization problem. As for every $u \in U_{ad}$ there is a unique $y \in H^1_0(\Omega)$ and $H^1_0(\Omega)$ is linear and continuously embedded in $L^2(\Omega)$, see [52, Chapter 2.5.1], it is convenient to define

$$\begin{aligned} S : L^2(\Omega) &\to L^2(\Omega), \\ u &\mapsto y \end{aligned}$$

as the *control-state operator*. With this definition, the optimal control problem (2.5) can be reformulated as quadratic optimization problem in the Hilbert space $L^2(\Omega)$

$$\min_{u\in U_{ad}} f(u) := \min_{u\in U_{ad}} \frac{1}{2}\|Su - y_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|u\|^2_{L^2(\Omega)}.$$

The function f is called *reduced objective*. Working with this reduced problem is called *black-box method*, which means exploiting the implicit function theorem and consider the state variable y as a function depending on the control u, i.e. $y = y(u)$. For this reduced problem, existence of an optimal solution can be proven.

THEOREM 2.42 (Existence and uniqueness)**.**
Let $(U, (\cdot,\cdot)_U)$ and $(V, (\cdot,\cdot)_V)$ be real Hilbert spaces, $U_{ad} \subset U$ a nonempty, closed and convex set, $y_d \in V$ and $\alpha \geq 0$. Furthermore let $S : U \to V$ be a linear and continuous operator. Then the quadratic optimization problems

$$\min_{u\in U_{ad}} f(u) := \frac{1}{2}\|Su - y_d\|^2_V + \frac{\alpha}{2}\|u\|^2_U$$

has an optimal solution $\bar{u}$. For $\alpha > 0$, this solution is unique.

Proof. [52, Theorem 2.14] □

An application of the Theorem 2.42 to the optimal control problem in Definition 2.39 establishes the following lemma.

LEMMA 2.43.
Under the assumptions made in Definition 2.39, the optimal control problem (2.5) has an optimal solution $\bar{u}$. For $\alpha > 0$, this solution is unique.

Proof. Set $U = H = L^2(\Omega)$ and $S = E_Y G$, where $E_Y G : H^1(\Omega) \to L^2(\Omega)$ denotes the embedding operator assigning a certain function in $L^2(\Omega)$ to every $y \in H^1(\Omega)$, as introduced in [52, Chapter 2.5.1]. The set $U_{ad} = \{u \in L^2(\Omega) : u_a \leq u \leq u_b\}$ is bounded, convex and closed. An application of Theorem 2.42 yields the assertion. □

For the case $\alpha = 0$, the solution is still unique because of the strict convexity of the functional $\|Su\|^2$ and U_{ad} being a convex set, see [7, Theorem 2.4]. One or both of the restrictions in U_{ad} can be omitted.

LEMMA 2.44.
If $\alpha > 0$, the optimal control problem in Definition 2.39 has an optimal solution if U_{ad} is only convex and closed. This covers, in particular, the cases $u_a = -\infty$ and/or $u_b = +\infty$.

Proof. [52, Theorem 2.16]. □

2.5 Characterization of an Optimal Solution

To characterize the optimal solution of the optimal control problem, some preliminary work is required. Starting with a generalization of the well-known derivative for operators in function spaces, in particular in Banach spaces, followed by the introduction of the adjoint operator, the optimality system is derived.

DEFINITION 2.45 (Derivatives)**.**
Let $F : U \to V$ be an operator mapping between real Banach spaces $(U, \|\cdot\|_U), (V, \|\cdot\|_V)$, $t \in \mathbb{R}$.

1. F is called *directionally differentiable at* $u \in U$ if the limit

$$dF(u,h) = \lim_{t \downarrow 0} \frac{F(u+th) - F(u)}{t} \in V$$

   exists for all $h \in U$. In this case, $dF(u,h)$ is called *directional derivative of* F *in the direction* h.

2. F is called *Gâteaux differentiable at* $u \in U$ if F is directional differentiable at u and the directional derivative $F' : U \to V, F'(u) : h \mapsto dF(u,h)$, is bounded and linear, i.e. $F'(u) \in L(U,V)$.

3. F is called *Fréchet differentiable at* $u \in U$ if F is Gâteaux differentiable at u and if the following approximation condition holds:

$$\frac{\|F(u+h) - F(u) - F'(u)h\|_V}{\|h\|_U} \to 0 \text{ for } \|h\|_U \to 0.$$

4. If F is Fréchet/Gâteaux differentiable in a neighborhood U_ϵ of $u \in U$ and $F' : U_\epsilon \to L(U,V)$ is itself Fréchet/Gâteaux differentiable at u, the mapping operator F is called *twice Fréchet/Gâteaux differentiable at* u, written $F''(x) \in L(U, L(U,V))$ for the second derivative of F in u. Analogously, higher derivatives can be defined.

Every Fréchet differentiable function F is Gâteaux differentiable (by definition) and these derivatives can be computed as directional derivatives, i.e. $F'(u)$ is the linear operator

$$F'(u) : h \mapsto \left.\frac{d}{dt}\right|_{t=0} F(u+th) \quad \forall h \in U.$$

For Fréchet differentiable operators, the so called *chain rule* holds.

LEMMA 2.46 (Chain rule)**.**
Let $(U, \|\cdot\|_U), (V, \|\cdot\|_V), (W, \|\cdot\|_W)$ be real Banach spaces and $F : U \to V$, $G : V \to W$ be Fréchet differentiable at $u \in U$ resp. at $F(u) \in V$. Then the operator

$$E(u) := G(F(u))$$

is Fréchet differentiable at u and

$$E'(u) = G'(F(u)) \cdot F'(u).$$

This holds also for Gâteaux differentiable functions and, as a direct consequence of the chain rule, also the *sum rule* holds. To obtain a concrete representation of the concept 'derivative', the representation of the gradient is given by

DEFINITION 2.47 (Gradient)**.**
Let $(U, (\cdot,\cdot)_U)$ be a real Hilbert space and $F : U \to \mathbb{R}$ a Fréchet differentiable function. Then the *gradient* ∇ is defined as the Riesz representation of the derivative F', such that

$$(\nabla F, u)_U = F'(u) \quad \forall u \in U.$$

With this definition, the representation of the derivative depends on the specific scalar product in U. Analogously to the definition of the gradient, the Hessian operator is defined.

DEFINITION 2.48 (Hessian)**.**
Let $(U, (\cdot,\cdot)_U)$ be a real Hilbert space and $F : U \to \mathbb{R}$ be a function which is twice Fréchet differentiable function in $u \in U$. Let the second derivative as a symmetric bilinear form be denoted by

$$D^2F(u)[h_1, h_2] := \frac{d}{dt_1}\bigg|_{t_1=0} \frac{d}{dt_2}\bigg|_{t_2=0} F(u + t_1h_1 + t_2h_2) \quad \forall h_1, h_2 \in U.$$

Then for each $h_1 \in U$ there exists a vector $v(h_1) \in U$ which is the Riesz representation of the linear form $D^2F(u)[h_1, \cdot]$ such that

$$D^2F(u)[h_1, h_2] = (v(h_1), h_2) \quad \forall h_2 \in U.$$

The linear mapping

$$\begin{aligned} \operatorname{Hess} F(u) : U &\to U \\ h_1 &\mapsto v(h_1) \end{aligned}$$

is called *Hessian operator at* u.

The aim of this section is to derive an optimality system characterizing the optimal solution of the optimal control problem. For this purpose, the adjoint operator is defined.

DEFINITION 2.49 (Adjoint operator)**.**
Let $(U, (\cdot,\cdot)_U)$ and $(V, (\cdot,\cdot)_V)$ be real Hilbert spaces and $A : U \to V$ a linear and continuous operator. Then A^* is called *adjoint operator* of A, if

$$(v, Au)_V = (A^*v, u)_U \quad \forall v \in V, u \in U$$

holds. A operator A is called *self-adjoint* if $A^* = A$.

The following proposition is the key for the derivation of first order necessary optimality conditions for optimal control problems with control constraints. To this end, several approaches exist, like the formal Lagrange approach in [7],[34] and [52] and the subsequent variational approach. Both methods involve the gradient of the reduced functional. There are two main possibilities to compute the derivative: the sensitivity approach [34, Chapter 1.6.1] and the adjoint approach, which is applied here.

LEMMA 2.50 (Necessary first order conditions)**.**
Let $(U, \|\cdot\|_U)$ be a real Banach space, $C \subset U$ convex and $f : C \to \mathbb{R}$ Gâteaux differentiable on C. Let $\bar{u} \in C$ be a solution of

$$\min_{u \in C} f(u).$$

Then the following variational inequality holds

$$f'(\bar{u})(u - \bar{u}) \geq 0 \quad \forall u \in C. \tag{2.6}$$

Proof. [52, Lemma 2.20] □

In case that f is convex, this first order necessary condition is also sufficient.

LEMMA 2.51 (Sufficient and necessary first order condition)**.**
Let $(U, \|\cdot\|_U)$ be a real Banach space, $C \subset U$ convex and $f : C \to \mathbb{R}$ Gâteaux differentiable on C and convex. If $\bar{u}$ is a solution of the variational inequality (2.6), then $\bar{u}$ also is a solution of

$$\min_{u \in C} f(u).$$

Proof. [52, Lemma 2.21] □

To apply the previous Lemma 2.51 for deriving the optimality conditions, the gradient of the reduced functional is needed. For

$$\min_{u \in U_{ad}} f(u) := \frac{1}{2}\|Su - y_d\|_U^2 + \frac{\alpha}{2}\|u\|_U^2$$

the Gâteaux derivative can be determined:

1. Consider $f_1(u) := \frac{1}{2}\|Su - y_d\|_U^2$. Then f_1 is a composed function, $f_1(u) = G(F(u))$ with $G(v) = \|v\|_U^2$ and $F(u) = Su - y_d$. Application of the chain rule

$$\begin{aligned} f_1'(u)h &= (v, F'(u)h)_U \\ &= (Su - y_d, Sh)_U \\ &= (S^*(Su - y_d), h)_U \end{aligned}$$

   and thus

$$f_1'(u)h = (S^*(Su - y_d), h)_U.$$

   Here $S^* \in L(U, U)$ is the adjoint operator of S.

2. Consider $f_2(u) := \frac{\alpha}{2}\|u\|_U^2$. Then

$$\begin{aligned} \lim_{t \to \infty} \frac{1}{t}(f_2(u + th) - f_2(u)) &= \lim_{t \to \infty} \frac{1}{t}(\frac{\alpha}{2}\|u + th\|_U^2 - \frac{\alpha}{2}\|u\|_U^2) \\ &= \lim_{t \to \infty} \frac{1}{t}(\alpha t(u, h)_U + \alpha t^2 \|h\|_U^2) \\ &= (\alpha u, h)_U \end{aligned}$$

   and thus

$$f_2'(u)h = (\alpha u, h)_U.$$

3. Consider $f(u) = f_1(u) + f_2(u)$ and apply the sum rule, this proves that the gradient has the form
$$f'(u) = f_1'(u) + f_2'(u) = S^*(Su - y_d) + \alpha u. \tag{2.7}$$

Altogether this leads to the quantification of an optimal solution of the optimal control problem.

THEOREM 2.52.
Let $(U, (\cdot,\cdot)_U)$, $(V, (\cdot,\cdot)_V)$ be real Hilbert spaces, $U_{ad} \subset U$ nonempty, closed and convex, $y_d \in V$, $\alpha \geq 0$ and $S : U \to V$ be a linear and continuous operator. Then $\bar{u} \in U_{ad}$ is solution of (2.4) if and only if $\bar{u}$ satisfies the variational inequality

$$(S^*(S\bar{u} - y_d) + \alpha\bar{u}, u - \bar{u})_U \geq 0 \quad \forall u \in U_{ad}$$

Proof. Application of the previous Lemma 2.51 and the calculation of the gradient (2.7). □

Now the last step is to determine the adjoint operator S^*.

LEMMA 2.53.
The adjoint operator $S^* : L^2(\Omega) \to L^2(\Omega)$ for the general second order elliptic operator with Dirichlet boundaries defined in (2.5) is given by

$$S^*z = p$$

where $p \in H_0^1(\Omega)$ is the weak solution of

$$\begin{aligned} Lp &= z \quad \text{in } \Omega \\ p &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

Proof. [52, Lemmata 2.23, 2.24] □

The variational inequality in Lemma (2.6) is now reformulated.

DEFINITION 2.54 (Adjoint equation)**.**
The weak solution $p \in H_0^1(\Omega)$ of the adjoint equation

$$\begin{aligned} Lp &= \bar{y} - y_d \quad \text{in } \Omega \\ p &= 0 \qquad\quad \text{on } \partial\Omega \end{aligned} \tag{2.8}$$

is called the adjoint state of y.

The adjoint equation is uniquely solvable, as $y_d \in L^2(\Omega)$ and $y \in H_0^1(\Omega)$, and $H_0^1(\Omega)$ can continuously be embedded in $L^2(\Omega)$. Hence due to Theorem 2.38, the adjoint equation (2.8) has a unique solution. For $z = \bar{y} - y_d$ in the previous lemma, one obtains

$$S^*(S\bar{u} - y_d) = S^*(\bar{y} - y_d) = p$$

and

$$(p + \alpha\bar{u}, u - \bar{u})_{L^2(\Omega)} \geq 0 \quad \forall u \in U_{ad}.$$

Altogether, the optimal solution of the optimal control problem can be characterized in the following theorem.

THEOREM 2.55.
Let $\bar{u}$ be the optimal control of (2.5) and $\bar{y}$ the appropriate state. There exists a weak solution p of the adjoint equation, which fulfills the variational inequality

$$\int_\Omega (p(x) + \alpha\bar{u}(x))(u(x) - \bar{u}(x))\,\mathrm{dx} \geq 0 \quad \forall u \in U_{ad}.$$

Vice versa, every $\bar{u} \in U_{ad}$ which satisfies the variational inequality with its appropriate state $\bar{y}$ and solution p of the adjoint equation is optimal.

Proof. [52, Theorem 2.25] □

Thus a control u is optimal, if and only if the tripple (y, u, p) fulfills the optimality system:

state equation

$$\begin{aligned} Ly &= f + u \quad \text{in } \Omega \\ y &= 0 \qquad \text{on } \partial\Omega \end{aligned} \tag{2.9}$$

adjoint equation

$$\begin{aligned} Lp &= y - y_d \quad \text{in } \Omega \\ p &= 0 \qquad \text{on } \partial\Omega \end{aligned} \tag{2.10}$$

design equation

$$(p + \alpha u, v - u)_{L^2(\Omega)} \geq 0 \quad \forall v \in U_{ad}. \tag{2.11}$$

For $U_{ad} = L^2(\Omega)$, the design equation simplifies to

$$p + \alpha u = 0.$$

This proves the existence and uniqueness of an optimal solution of the model problem and characterizes this optimal solution and finishes the first chapter.

REMARK 2.56.
The reduced Hessian, i.e. the Hessian of the reduced operator in the $L^2(\Omega)$ scalar product is

$$\mathrm{Hess}\, f(u) = L^{-2} + \alpha I.$$

L^{-2} is a compact operator and in general not coercive, so that a system solution with this operator will be ill-posed. The reduced Hessian is a compact perturbation of the operator αI and the complete reduced Hessian is coercive. This structure of the reduced Hessian operator is typical in PDE-constrained optimization problems and shows that in general regularization ($\alpha > 0$) is needed to have a well-posed problem formulation. For more details, see [7, Example 2.16(c)]. In [54], the eigenvalues of $\mathrm{Hess}\, f(u)$ are given by

$$\lambda_{mn} = \alpha + \frac{1}{\pi^4(m^2 + n^2)^2}, \quad m, n \in \mathbb{N}$$

and as $\lambda_{mn} > 0$, $\mathrm{Hess}\, f(u)$ is strictly positive definite.

3 Finite Difference Multigrid Solver

The generic setting of PDE-based optimization and in particular optimal control problems is naturally a infinite-dimensional Hilbert space. Solving these optimization problems numerically requires approximating these functional spaces by finite-dimensional spaces. In general, there are two different approaches for this approximation: '*first discretize, then optimize*' and '*first optimize, then discretize*'. On the one side, *first discretize, then optimize* means a discretization of the underlying PDE and the objective. Afterwards the optimality condition, resulting in a finite-dimensional optimization problem, is derived. On the other side, *first optimize, then discretize* means that the optimality system is derived first, i.e. in function spaces. In this approach, the discretization takes place in the second step. For more details about these two different approaches see [7, Chapter 3.1] and [34, Chapter 3]. This thesis deals with the second approach. In Section 3.1 the optimality system is reformulated for the case that there are no restrictions to the control. Section 3.2 gives an analysis of regularity and consistency for the second order finite difference discretization of the PDE and the resulting optimality system, which characterizes the optimal solution of the optimal control problem. There are several optimization methods for solving the optimal control problem: an application of the implicit function theorem leads to so-called *black box* approaches where the PDE-constrained optimization problem is transformed into an unconstrained optimization problem and the reduced gradient for these reduced functional is computed via the adjoint approach. Gradient-based optimization schemes like steepest descent, Quasi-Newton methods which approximate the Hessian by low-rank updates based on gradient evaluations or Krylov-Newton methods can be applied. Semi-smooth Newton methods for the control-constrained formulation and (reduced) SQP methods are proposed in [7, Chapter 4]. These methods are called *one-shot methods*, representing a solution method for optimization problems which solves the optimization problem during the solution of the state equation. Another important approach for solving optimal control problems are multigrid methods. Originally these are a class of algorithms for solving systems arising from the discretization of partial differential equations. A general introduction into linear geometric multigrid methods is given in Section 3.3. For optimization purposes, one can distinguish between a direct multigrid approach on the one hand, where the optimization problem is implemented within the grid hierarchy and on the other hand, there is an approach using a multigrid scheme as inner solver within an outer optimization loop. As representative of the first kind, the *collective smoothing multigrid (CSMG)*, proposed in [7], is considered in Section 3.4 for solving the discretized optimality system.

3.1 Reformulation of the Optimality System

Recalling the academic elliptic optimal control problem in Definition 2.39

$$\min_{(y,u)} J(y,u) := \min_{(y,u)} \frac{1}{2}\|y - y_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|u\|^2_{L^2(\Omega)}$$

$$\begin{aligned} Ly &= f + u \quad \text{in } \Omega \\ y &= 0 \qquad \text{on } \partial\Omega \end{aligned}$$

and the characterization of an optimal solution by the optimality system (2.9)-(2.11)

$$\begin{aligned} Ly &= f + u \quad && \text{in } \Omega \\ y &= 0 && \text{on } \partial\Omega \\ L\tilde{p} &= y - y_d && \text{in } \Omega \\ \tilde{p} &= 0 && \text{on } \partial\Omega \\ \alpha u + \tilde{p} &= 0 && \text{in } \Omega. \end{aligned}$$

Assuming $\alpha > 0$, the control u can be eliminated by reformulating the design equation

$$\alpha u + \tilde{p} = 0 \Leftrightarrow u = -\frac{\tilde{p}}{\alpha} \tag{3.1}$$

and by substituting $p := -\tilde{p}$, the optimality system can be written as coupled system of two infinite-dimensional operators

$$\begin{bmatrix} \alpha L & -I \\ I & L \end{bmatrix} \begin{pmatrix} y \\ p \end{pmatrix} = \begin{pmatrix} \alpha f \\ y_d \end{pmatrix} \tag{3.2}$$

where L is understood with homogeneous Dirichlet boundary conditions. To simplify notation, the following abbreviation is used for the optimality system

$$\mathcal{A} := \begin{bmatrix} \alpha L & -I \\ I & L \end{bmatrix}, \quad w := \begin{pmatrix} y \\ p \end{pmatrix} \quad \text{and} \quad \phi := \begin{pmatrix} \alpha f \\ y_d \end{pmatrix} \tag{3.3}$$

and the problem formulation is given by

$$\mathcal{A}w = \phi.$$

To give some statements about accuracy and convergence properties, stronger assumptions with regard to the regularity of the solution of the adjoint and design equation are needed, see [30, Chapter 9.1]. For the case of the academic model problem, this regularity is at hand. Let L be the uniformly elliptic and symmetric operator defined in Definition 2.29.

LEMMA 3.1.
If $y_d, f \in L^2(\Omega)$, $\Omega \in C^{1,1}$ bounded and $a_{ij} \in C^{0,1}(\Omega)$, then the solution of the optimality system fulfills $(\bar{y}^*, \bar{u}^*, \bar{p}^*) \in (H^1_0(\Omega) \cap H^2(\Omega))^3$.

Proof. [26, Theorem 2.2.2.3] □

Even weaker assumption to the area Ω are possible:

LEMMA 3.2.
Let Ω be bounded and convex, $y_d, f \in L^2(\Omega)$ and $a_{ij} \in C^{0,1}(\Omega)$. Then $(\bar{y}^*, \bar{u}^*, \bar{p}^*) \in (H_0^1(\Omega) \cap H^2(\Omega))^3$.

Proof. [26, Theorem 3.2.1.2, Theorem 3.2.1.3] □

3.2 Discretization of the Optimality System

In this section the discretization of the optimality system with the finite differences method is investigated and regularity as well as consistency of the solution of this discretized system to the solution of the continuous coupled system $\mathcal{A}w = \phi$ is proven.

3.2.1 Finite Difference Discretization

The second order elliptic differential operator and the optimality system are discretized by means of a finite difference approximation of second order. To apply this method, the domain Ω, where the PDE takes effect on, is covered with a finite number of grid points and the derivatives at these grid points are replaced by approximate finite differences. In the following the notation and terminology of [30] is used.

DEFINITION 3.3 (Grid)**.**
For $\Omega \subset \mathbb{R}^d$ and a grid size $h > 0$ define a infinite grid

$$Q_h := \{x \in \mathbb{R}^d : x_i = s_i h, s_i \in \mathbb{Z}^d\}$$

and

$$\Omega_h := \Omega \cap Q_h.$$

Ω_h is called uniform grid on Ω.

On a grid so-called grid functions are defined.

DEFINITION 3.4 (Grid function)**.**
Let Ω_h be a grid on $\Omega \subset \mathbb{R}^d$.

1. A function $u_h : \Omega_h \to \mathbb{R}$ is called grid function.
2. $U_h := \{u_h : \Omega_h \to \mathbb{R}\}$ is the linear space of grid functions.
3. A grid function $u_h \in U_h$ is extended on Q_h by setting $u_h(x) = 0$ for $x \in Q_h \backslash \Omega_h$.

The derivatives emerging in the differential operator are replaced by approximative finite differences at each grid point.

DEFINITION 3.5 (Difference quotient)**.**
Let $u_h \in U_h$, $h > 0$ be the grid size, e_i be the i-th unit vector and $x \in \mathbb{R}^d$.

1. First order forward space difference quotient of u_h in the x_i direction:
$$D^+_{x_i} u_h(x) = \frac{u_h(x + he_i) - u_h(x)}{h}.$$
2. First order backward space difference quotient of u_h in the x_i direction:
$$D^-_{x_i} u_h(x) = \frac{u_h(x) - u_h(x - he_i)}{h}.$$
3. First order central space difference quotient of u_h in the x_i direction:
$$D^0_{x_i} u_h(x) = \frac{1}{2}(D^+_{x_i} u_h(x) + D^-_{x_i} u_h(x)).$$
4. Second order derivatives are approximated by:
$$D^-_{x_i} D^+_{x_i} u_h(x) = \frac{1}{h^2}\left[u_h(x + he_i) - 2u_h(x) + u_h(x - he_i)\right].$$

In analogy to the infinite-dimensional spaces $L^2(\Omega)$ and $H^1(\Omega)$, the discrete spaces L^2_h and H^1_h of grid functions are introduced. The suitable norms are given in the following definitions.

DEFINITION 3.6 (Discrete L^2-norm)**.**
Let $\Omega \subset \mathbb{R}^d$ and $u_h, v_h \in U_h$. The discrete L^2-scalar product is defined by
$$(u_h, v_h)_0 := (u_h, v_h)_{L^2_h} := h^d \sum_{x \in Q_h} u_h(x) v_h(x)$$
and the associated discrete L^2-norm is given by
$$|u_h|_0 := \|u_h\|_{L^2_h} := \left(h^d \sum_{x \in Q_h} (u_h(x))^2 \right)^{1/2}.$$
Also define
$$|u_h|_\infty := \max_{x \in Q_h} |u_h(x)|.$$

DEFINITION 3.7 (Discrete H^1-norm)**.**
Let $u_h, v_h \in U_h$. The discrete H^1-norm is given by
$$|u_h|_1 := \|u_h\|_{H^1_h} := \left(|u_h|_0^2 + \sum_{i=1}^{n} |D^-_j u_h|_0^2 \right)^{1/2}.$$
The dual norm of $|\cdot|_1$ is defined by
$$|u_h|_{-1} := \|u_h\|_{H^{-1}_h} := \sup \left\{ \frac{|(u_h, v_h)_0|}{|v_h|_1} : v_h(x) \neq 0, v_h(x) = 0 \text{ for } x \in Q_h \backslash \Omega_h \right\}.$$

DEFINITION 3.8 (Matrix norm)**.**
Let U_h, V_h be discrete linear spaces with vector norms $\|\cdot\|_{U_h}, \|\cdot\|_{V_h}$ as well as $u_h \in U_h$ and $L_h : U_h \to V_h$. Then

$$\|L_h\|_{L_h(U_h,V_h)} := \sup\left\{\frac{\|L_h u_h\|_{V_h}}{\|u_h\|_{U_h}} : u_h(x) \neq 0, u_h(x) = 0 \text{ for } x \in Q_h \backslash \Omega_h\right\}$$

is called the matrix norm associated to the vector norms $\|\cdot\|_{U_h}$ and $\|\cdot\|_{V_h}$.

The discrete H^2-norm $\|\cdot\|_{H_h^2}$ is defined in [30, Chapter 9.2.4].

DEFINITION 3.9.
For $u_h, v_h \in U_h$ define

$$V_h := \{u_h \in U_h : u_h = 0 \text{ on } \partial\Omega_h\}$$

and for the system of nodal functions $(u_h, v_h) \in V_h \times V_h$ set $V_h^2 := V_h \times V_h$.

A discrete version of the Poincaré-Friedrichs inequality holds.

LEMMA 3.10 (Discrete Poincaré-Friedrichs inequality)**.**
Let $\Omega \subset \mathbb{R}^d$. For any grid function $v_h \in V_h$ there exists a constant $\bar{c} > 0$ such that

$$|v_h|_0^2 \leq \bar{c} \sum_{i=1}^{d} |D_i^- v_h|_0^2,$$

where $\bar{c}$ is independent of v_h and h.

Proof. [27, Lemma 2.8] □

The constant only depends on the area Ω. For a particular $\Omega = (a, b) \times (c, d)$, $\bar{c}$ can be determined, see [31] or [28], by

$$\bar{c} = \left(\frac{2}{(b-a)^2} + \frac{2}{(d-c)^2}\right)^{-1}$$

and thus for $\Omega = (0, 1) \times (0, 1)$, $\bar{c} = \frac{1}{4}$.

DEFINITION 3.11 (Restriction operators)**.**

1. For $u \in L^2(\Omega)$ and $v \in H^2(\Omega)$ define a restriction operator

$$\begin{aligned} \tilde{R}_h : L^2(\Omega) &\to L_h^2 \\ u &\mapsto u_h \end{aligned}$$

and

$$\begin{aligned} R_h : H_0^1(\Omega) \cap H^2(\Omega) &\to H_h^2 \\ v &\mapsto v_h, \end{aligned}$$

where u, v are approximated by the grid functions u_h, v_h through the mean values with respect to elementary cells depending on the dimension, given by

(1D) $[x - \frac{h}{2}, x + \frac{h}{2}]$:

$$u_h(x) = h \int_{x-\frac{h}{2}}^{x+\frac{h}{2}} u(x)\, \mathrm{dx}$$

(2D) $[x_1 - \frac{h}{2}, x_1 + \frac{h}{2}] \times [x_2 - \frac{h}{2}, x_2 + \frac{h}{2}]$:

$$u_h(x_1, x_2) = h^2 \int_{x_1-\frac{h}{2}}^{x_1+\frac{h}{2}} \int_{x_2-\frac{h}{2}}^{x_2+\frac{h}{2}} u(x_1, x_2)\, \mathrm{d}x_1 \mathrm{d}x_2$$

(3D) $[x_1 - \frac{h}{2}, x_1 + \frac{h}{2}] \times [x_2 - \frac{h}{2}, x_2 + \frac{h}{2}] \times [x_3 - \frac{h}{2}, x_3 + \frac{h}{2}]$:

$$u_h(x_1, x_2, x_3) = h^3 \int_{x_1-\frac{h}{2}}^{x_1+\frac{h}{2}} \int_{x_2-\frac{h}{2}}^{x_2+\frac{h}{2}} \int_{x_3-\frac{h}{2}}^{x_3+\frac{h}{2}} u(x_1, x_2, x_3)\, \mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}x_3.$$

2. Define

$$\tilde{R}_h^2 : L^2(\Omega) \times L^2(\Omega) \to L_h^2 \times L_h^2 \quad \text{and}$$
$$R_h^2 : (H_0^1(\Omega) \cap H^2(\Omega)) \times (H_0^1(\Omega) \cap H^2(\Omega)) \to H_h^2 \times H_h^2$$

by

$$\tilde{R}_h^2 = (\tilde{R}_h, \tilde{R}_h) \text{ resp. } R_h^2 = (R_h, R_h).$$

3. Continuous functions are represented by their grid values:
For $u \in C^k(\bar{\Omega})$, $k = 0, 1, \ldots$, the restriction operator $\bar{R}$ on $\bar{\Omega}_h$ is denoted by

$$(\bar{R}_h u)(x) = u(x).$$

For the purpose of simplification and to avoid technical difficulties, the domain $\Omega \subset \mathbb{R}^2$ is assumed to be rectangular and the grid size h chosen such that the boundaries of Ω coincide with grid lines. Furthermore, let $a_{ik} = a_{ki} = 0$ for $i \neq k$, $i, k = 1, 2$, i.e. the differential operator

$$L = -\operatorname{div}(A \operatorname{grad}) + c$$

applied to u states

$$\begin{aligned} Lu = &- \frac{\partial}{\partial x_1}\left(a_{11}(x_1, x_2)\frac{\partial}{\partial x_1}u(x_1, x_2)\right) - \frac{\partial}{\partial x_2}\left(a_{22}(x_1, x_2)\frac{\partial}{\partial x_2}u(x_1, x_2)\right) \\ &+ c(x_1, x_2)u(x_1, x_2). \end{aligned}$$

The second order finite difference approximation of L (with an additional treatment of the homogenous Dirichlet boundaries) is given by the five-point scheme

$$\begin{aligned} L_h u_h(x_1, x_2) = &- D_{x_1}^-(a_{11}(x_1, x_2)D_{x_1}^+ u_h(x_1, x_2)) - D_{x_2}^-(a_{22}(x_1, x_2)D_{x_2}^+ u_h(x_1, x_2)) \\ &+ c(x_1, x_2)u_h(x_1, x_2) \end{aligned}$$

in $(x_1, x_2) \in \Omega_h$. As L_h is assumed to be uniformly elliptic, L_h is H_h^1-regular and consistent with L from $H^2(\Omega)$ to H_h^{-1}. For the definitions of H_h^1-regularity, consistency and the proof of these statements see [30, Example 9.2.5, Theorem 9.2.14].

REMARK 3.12.

1. The discretization of L is a second order consistency scheme, i.e.

$$|L_h \bar{R}_h u - \bar{R}_h L u|_\infty \leq c h^2 \|u\|_{C^4(\bar{\Omega})},$$

   see [30, Remark 5.1.13].

2. In case of more general convex domains Ω or if the boundary does not coincide with grid lines, special attention must be paid to the discretization along the boundaries, Shortley-Weller discretization, see [30, Lemma 9.2.7].

3. For more general differential operators, in particular the case $a_{ik} \neq 0$ for $i \neq k$, compare to the nine-point discretization in [30, Chapter 5.1.4].

With these definitions and simplifications the discrete optimal control problem is specified by

$$\begin{aligned} \min_{(y_h,u_h)} J(y_h, u_h) := \min_{(y_h,u_h)} \frac{1}{2}|y_h - \tilde{R}_h y_d|_0^2 + \frac{\alpha}{2}|u_h|_0^2 \\ L_h y_h = u_h + \tilde{R}_h f, \quad u_h \in L_h^2 \end{aligned}$$

and the discretized optimality system after eliminating the control u has the form

$$\begin{aligned} \alpha L_h y_h - p_h &= \alpha \tilde{R}_h f \\ L_h p_h + y_h &= \tilde{R}_h y_d. \end{aligned} \tag{3.4}$$

The infinite-dimensional operator $\mathcal{A}$ in (3.3) is well-defined on

$$H_0^1(\Omega) \times H_0^1(\Omega) \to H^{-1}(\Omega) \times H^{-1}(\Omega)$$

as well as on

$$(H^2(\Omega) \cap H_0^1(\Omega)) \times (H^2(\Omega) \cap H_0^1(\Omega)) \to L^2(\Omega) \times L^2(\Omega).$$

Now the family of operators

$$\mathcal{A}_h = \begin{bmatrix} \alpha L_h & -I_h \\ I_h & L_h \end{bmatrix}, \tag{3.5}$$

where I_h is the identity operator on grid functions $v_h \in V_h$ is defined. The aim is to show the convergence of a solution of the discretized system (3.4) to the solution of $\mathcal{A}w = \phi$ for $h \to 0^+$. The operators $\mathcal{A}_h$ are defined between product spaces of grid functions

$$\begin{aligned} &\mathcal{A}_h : H_h^1 \times H_h^1 \to H_h^{-1} \times H_h^{-1} \quad \text{and} \\ &\mathcal{A}_h : H_h^2 \times H_h^2 \to L_h^2 \times L_h^2. \end{aligned}$$

3.2.2 Regularity and Consistency

For proving optimal error estimates, regularity of the discretization as well as consistency of the discretized operators to the continuous analogon have to be shown.

DEFINITION 3.13.
A family $\{\mathcal{A}_h\}_{h>0}$ is called

1. H_h^1-regular, if $\mathcal{A}_h$ is invertible and there exists a constant C_1 independent of h such that
$$\|\mathcal{A}_h^{-1}\|_{L(H_h^{-1}\times H_h^{-1}, H_h^1\times H_h^1)} \leq C_1.$$
2. H_h^2-regular, if $\mathcal{A}_h$ is invertible and there exists a constant C_2 independent of h such that
$$\|\mathcal{A}_h^{-1}\|_{L(L_h^2\times L_h^2, H_h^2\times H_h^2)} \leq C_2.$$

The following lemma shows H_h^1-regularity, including the existence of $\mathcal{A}_h^{-1}$, by applying the Lax-Milgram Lemma.

LEMMA 3.14 (Regularity)**.**
The family of operators $\{A_h\}_{h>0}$ with h such that the boundaries of Ω are grid lines is H_h^1-regular.

Proof. Straightforward extension of [5, Lemma 3.2] for more general second order differential operators. □

The following consistency result holds.

LEMMA 3.15 (Consistency)**.**
There exists a constant C_K independent of h such that
$$\|\mathcal{A}_h R_h^2 - \tilde{R}_h^2 \mathcal{A}\|_{L((H^2\cap H_0^1)^2, (H_h^{-1}\times H_h^{-1}))} \leq C_K h.$$

Proof. [5, Lemma 3.3] □

As $\mathcal{A}_h$ is H_h^1-regular and consistent with $\mathcal{A}$ from $(H^2(\Omega)\cap H_0^1(\Omega))\times(H^2(\Omega)\cap H_0^1(\Omega))$ to $H_h^{-1}\times H_h^{-1}$ shown in the preceding lemma, the following two convergence statements count.

THEOREM 3.16 (H_h^1-convergence)**.**
There exists a constant K_1, depending on Ω, f, y_d, but independent of h, such that
$$|y_h - R_h y|_1 + |u_h - R_h u|_1 + |p_h - R_h p|_1 \leq K_1 h.$$

Proof. [5, Lemma 3.4] □

THEOREM 3.17 (L_h^2-convergence)**.**
Let the boundaries of Ω be grid lines. There exists a constant K_2, depending on Ω, f, y_d but independent of h, such that
$$|y_h - R_h y|_0 + |u_h - R_h u|_0 + |p_h - R_h p|_0 \leq K_2 h^2.$$

Proof. [5, Theorem 3.5] □

REMARK 3.18.
For higher-order finite difference approximations of the elliptic optimality system, see e.g. [7, Chapter 3.2] and the literature given therein.

3.3 Multigrid Methods

In general, *geometric multigrid methods (GMG)* are a class of algorithms for solving systems of linear equations arising from the discretization of partial differential equations. In contrast, *algebraic multigrid methods (AMG)* do not require a given problem to be defined on a grid but rather operates directly on algebraic equations [53, Appendix A]. The main advantages of the linear multigrid method are the optimal computational complexity, i.e. the number of required computer iterations scales linearly with the number of unknowns and the rate of convergence, which is independent of the grid size h. The convergence speed does not deteriorate when the discretization is refined and for that reason, an acceptable approximation of the infinite-dimensional problem can be achieved. Therefore, in many cases multigrid methods are amongst the fastest solution techniques. They can also directly be applied to more sophisticated non-symmetric problems or in the absence of positive definiteness. For solving non-linear problems, the *full approximation scheme (FAS)* is the most popular approach [6], [53, Chapter 5.2], [29, Chapter 9].

3.3.1 Geometric Multigrid Method for Linear Problems

In this section, a brief introduction into the general multigrid methodology and terminology is given. A more detailed survey can be found in [7, Chapter 5], [12], [29], [16] or [53]. To keep this more general, consider the discretization of a general scalar linear differential operator $Au = f$. As errors are affected with a large spectrum of frequencies, the multigrid method is based on two complementary schemes:

- The *high-frequency* components of the error are reduced and smoothed by applying iterative methods.
- The *low-frequency* error components are reduced by the coarse grid correction scheme.

To introduce the linear multigrid method, define a sequence of grid sizes $\{h_k\}_{k=0}^{\infty}$ generating a grid hierarchy $\Omega_k := \Omega_{h_k}$ on Ω. The operators and variables on a grid Ω_k are indexed with the level number k, $k = 0, \ldots, L$ where L denotes the finest level. On each level k, a problem

$$A_k u_k = f_k \tag{3.6}$$

has to be solved, where A_k arises from the discretization of the boundary value problem. Introducing the error grid function $e_k := u^* - u_k$, where u^* is the exact solution, and the defect $d_k := f_k - A_k u_k$, solving (3.6) is equivalent to solving

$$A_k e_k = d_k.$$

As it is assumed that u_k is affected with a large spectrum of errors, the aim of the multigrid strategy is to solve for all frequencies using multiple grids. To reduce the high-frequent errors, on every grid k a *smoothing procedure* is applied. A *smoother* is an iterative scheme applied ν_1-times denoted by

$$S_k^{\nu_1}(u_k, f_k)$$

and one step of this smoothing iteration is written in the form

$$u_k^{\nu_1} = u_k^{(\nu_1 - 1)} + R_k \left(f_k - A_k u_k^{(\nu_1 - 1)} \right)$$

where the operator R_k applies to the residual. Different choices of smoothers are discussed in the next section. The (low-frequency) smooth components of the error are reduced by the *coarse grid correction (CGC)*. For this purpose, a coarse grid problem is constructed on the coarser grid Ω_{k-1}. Define a restriction operator I_k^{k-1} mapping from the fine grid Ω_k to the coarser one Ω_{k-1} and maintain the coarse grid problem

$$A_{k-1}e_{k-1} = I_k^{k-1}d_k.$$

After solving for the error on the coarse grid, a prolongation operator I_{k-1}^k transfers the error from the coarse to the finer grid. Here, the current approximation at level k obtained by the smoothing process and before coarsening u_k is updated by

$$u_k^{new} = u_k + I_{k-1}^k e_{k-1}.$$

If the high-frequency components of the error on the finer grid k are well-damped, i.e. the error is sufficiently smooth, then the solution at level $k-1$ should provide a good approximation of the error e_k by $I_{k-1}^k e_{k-1}$.
The coarse grid correction is followed by ν_2 post-smoothing steps.

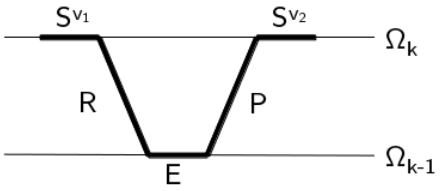


Figure 3.1: Two-grid iteration

In Figure 3.1 S^{ν_i} denotes ν_i smoothing steps on grid Ω_k, R the restriction operator I_k^{k-1}, P the prolongation operator I_{k-1}^k and E the exact solution for the error on the coarser grid Ω_{k-1}. This *two-grid scheme* is a linear iterative method.

LEMMA 3.19 (Two-grid iteration)**.**
The two-grid iteration matrix is given by

$$M_k^{TGM(\nu_1,\nu_2)} = S_k^{\nu_2}\left(I_k - I_{k-1}^k A_{k-1}^{-1} I_k^{k-1} A_k\right) S_k^{\nu_1},$$

where I_k is the identity matrix and S_k is the smoothing iteration matrix.

Proof. [40, Lemma 4.44] □

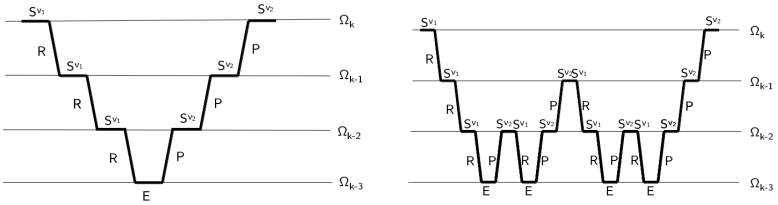
In the two-grid scheme the size of the coarse grid may still be too large to be solved exactly. The idea of transferring to a coarser grid can be applied along a set of nested meshes. Starting on level k with a given initial approximation and applying ν_1 smoothing steps, the residual is computed and transferred to the next coarser grid. This procedure is repeated until the coarsest grid is reached. Here the problem should be acceptably small and can be solved sufficiently exact by using a direct or iterative method. The result is used to improve u_k and followed by ν_2 post-smoothing steps at level k and the CGC procedure is repeated until the finest level is reached. The whole process represents one multigrid cycle, formulated in Algorithm 3.1.

Algorithm 3.1 Multigrid scheme

1: If $k = 0$: solve $A_0 u_0 = f_0$ exactly
2: Pre-smoothing:
$$u_k^{(l)} = S_k\left(u_k^{(l-1)}, f_k\right), \quad l = 1, \ldots \nu_1.$$
3: Compute the residual:
$$d_k = f_k - A_k u_k^{(\nu_1)}.$$
4: Restriction:
$$d_{k-1} = I_k^{k-1} d_k.$$
5: Set $u_{k-1} = 0$.
6: Call γ times the multigrid cycle to solve $A_{k-1} u_{k-1} = d_{k-1}$.
7: Coarse grid correction:
$$u_k^{(\nu_1+1)} = u_k^{(\nu_1)} + I_{k-1}^k u_{k-1}.$$
8: Post-smoothing:
$$u_k^{(l)} = S_k\left(u_k^{(l-1)}, f_k\right), \quad l = \nu_1 + 1, \ldots, \nu_1 + \nu_2 + 1.$$

The parameter γ is called the cycle index. For $\gamma = 1$, the multigrid cycle is called V-cycle and for $\gamma = 2$ W-cycle.

Both variants are illustrated in Figure 3.2.



(a) Multigrid V-cycle (b) Multigrid W-cycle

Figure 3.2: Multigrid iteration

The multigrid iteration matrix is given in recursive form.

LEMMA 3.20 (Multigrid iteration)**.**
For $k = 0$, let $M_0 = 0$. For $k = 1, \ldots, L$,

$$M_k = S_k^{\nu_2}\left(I_k - I_{k-1}^k\left(I_{k-1} - M_{k-1}^{\gamma} A_{k-1}^{-1} I_k^{k-1} A_k\right)\right) S_k^{\nu_1}.$$

where I_k is the identity matrix, S_k the smoothing iteration matrix and M_k is the multigrid iteration matrix for level k.

Proof. [7, Lemma 5.6] □

As there is no general multigrid algorithm working for all boundary value problems, various components have to be adapted to the special character of a particular problem.

3.3.2 Smoothing Procedure

Let A_k be a symmetric and positive definite matrix arising from the discretization of a boundary value problem on Ω_k, and f_k be the appropriate right-hand side. Most of the facts listed are also valid for more general matrices and not restricted to the discretization of a differential operator. The most common choice for solving the system $A_k u_k = f_k$ are linear iterative methods of the form

$$u_k^{(l+1)} = S_k\left(u_k^{(l)}, f_k\right),$$

where S_k can be represented as

$$S_k\left(u_k^{(l)}, f_k\right) = M_k u_k^{(l)} + N_k f_k. \tag{3.7}$$

M_k is called iteration matrix. This iteration matrix is the *amplification matrix* of the error, as for $e_k^{(l)} := u^* - u_k^{(l)}$, with $u^* = A_k^{-1} f_k$ the exact solution, the iteration (3.7) is equivalent to $e_k^{(l+1)} = M_k e_k^{(l)}$, see [29]. To ensure convergence, an additional assumption regarding to the eigenvalues of the matrix M_k is needed.

DEFINITION 3.21 (Eigenvector, eigenvalue, spectrum, spectral radius)**.**

1. A vector $x \in \mathbb{C}^n \backslash \{0\}$ is an *eigenvector* of the matrix $A \in \mathbb{C}^{n\times n}$ with *eigenvalue* $\lambda \in \mathbb{C}$ if the following equation holds:
$$Ax = \lambda x.$$
2. The *spectrum* of $A \in \mathbb{C}^{n\times n}$ is defined by
$$\sigma(A) = \{\lambda \in \mathbb{C} : \lambda \text{ eigenvalue of } A\}.$$
3. The *spectral radius* of $A \in \mathbb{C}^{n\times n}$ is defined by
$$\rho(A) = \max\{|\lambda| : \lambda \in \sigma(A)\}.$$

THEOREM 3.22.
A linear iterative method $u_k^{(l+1)} = M_k u_k^{(l)} + N_k f_k$ converges for all initial values $u_k^{(0)}$ to a optimal solution u^* if and only if the spectral radius of M_k fulfills

$$\rho(M_k) < 1.$$

Proof. [40, Theorem 4.5] □

However, in the multigrid context the smoothing properties are much more important than the convergence properties.
The main approach for constructing linear iterative schemes is the idea of splitting the matrix A_k into

$$A_k = B_k + (A_k - B_k).$$

For a regular matrix B_k, the system $A_k u_k = f_k$ can be reformulated as

$$u_k = B_k^{-1}(B_k - A_k)u_k + B_k^{-1} f_k$$

and thus a linear iterative method

$$u_k^{(l+1)} = M_k u_k^{(l)} + N_k f_k$$

can be defined, where $M_k = B_k^{-1}(B_k - A_k)$ and $N_k = B_k^{-1}$. The two most convenient and often very effective representatives for smoothing techniques are the Jacobi iteration and the Gauss Seidel iteration with several variations. For a matrix $A_k \in \mathbb{R}^{n\times n}$ let (omitting the index k if clear)

$$D_k = [d_{ij}]_{i,j=1,\dots,n},\ d_{ij} = \begin{cases} a_{ij}, & i = j \\ 0, & \text{else} \end{cases}$$

be the diagonal part of the matrix A_k, D_k is assumed to be invertible,

$$L_k = [l_{ij}]_{i,j=1,\dots,n},\ l_{ij} = \begin{cases} a_{ij}, & i > j \\ 0, & \text{else} \end{cases}$$

be the strictly lower triangular part of A_k and

$$R_k = [r_{ij}]_{i,j=1,\dots,n},\ r_{ij} = \begin{cases} a_{ij}, & i < j \\ 0, & \text{else} \end{cases}$$

be the strictly upper triangular part of A_k.

3.3.2.1 Jacobi Iteration

The idea of the *Jacobi iteration (JAC)* is to split $A_k = D_k + A_k - D_k$ and to define

$$\begin{aligned} u_k^{(l+1)} &= D_k^{-1}(D_k - A_k)u_k^{(l)} + D_k^{-1} f_k \\ \Leftrightarrow u_k^{(l+1)} &= u_k^{(l)} + D_k^{-1}\left(f_k - A_k u_k^{(l)}\right) \end{aligned}$$

respectively written in component notation (and again omitting k)

$$u_i^{(l+1)} = \frac{1}{a_{ii}}\left(f_i - \sum_{j=1, j\neq i}^{n} a_{ij} u_j^{(l)}\right) \quad \text{for } i = 1, \dots, n.$$

A variation is the *damped Jacobi iteration (ω-JAC)* with a relaxation parameter $0 < \omega \leq 1$ and

$$u_k^{(l+1)} = u_k^{(l)} + \omega D_k^{-1}\left(f_k - A_k u_k^{(l)}\right).$$

For convergence properties see [40, Theorem 4.11, Theorem 4.13]. The (JAC) and (ω-JAC) are independent of the ordering of the grid points in Ω_k. For a particular problem, the optimal ω can be determined by an analytical eigenvalue decomposition, if available, or by numerical tests.

3.3.2.2 Gauss-Seidel Iteration

For the *Gauss-Seidel iteration (GS)*, the matrix A_k is splitted into $A_k = D_k + L_k + R_k$ and thus a linear iteration scheme is defined by

$$u_k^{(l+1)} = -(D_k + L_k)^{-1} R_k u_k^{(l)} + (D_k + L_k)^{-1} f_k.$$

In componentwise notation, dropping k and assuming that u_j^{m+1} for $j = 1, \ldots, i-1$ is already computed, this reads

$$u_i^{(l+1)} = \frac{1}{a_{ii}} \left(f_i - \sum_{j=1}^{i-1} a_{ij} u_j^{(l+1)} - \sum_{j=i+1}^{n} a_{ij} u_j^{(l)} \right) \quad \text{for } i = 1, \ldots, n.$$

Convergence of this scheme is e.g. studied in [40, Theorem 4.16, Corollar 4.18].
The splitting $A_k = D_k + L_k + R_k$ is unique, but the representation of a discrete boundary value problems by a system of equations $A_k u_k = f_k$ is by no means unique. The construction of A_k depends on the numbering of the unknowns. In the two-dimensional case there are several possibilities to enumerate the grid points in Ω_k [29, p. 51]. Assume

$$\Omega_k \subset Q_k = \{(ih_k, jh_k), i, j \in \mathbb{Z}\}.$$

1. *Lexicographical ordering*:
   A grid point $(ih_k, jh_k) \in \Omega_k$ precedes another point $(kh_k, lh_k) \in \Omega_k$, if and only if

   $$j < l \text{ or } (j = l \text{ and } i < k).$$

2. *Red-black ordering (checkerboard ordering)*:
   The grid points in Ω_k are separated into red points

   $$\Omega_k^{red} = \{(ih_k, jh_k) \in \Omega_k : i + j \text{ even}\}$$

   and black points

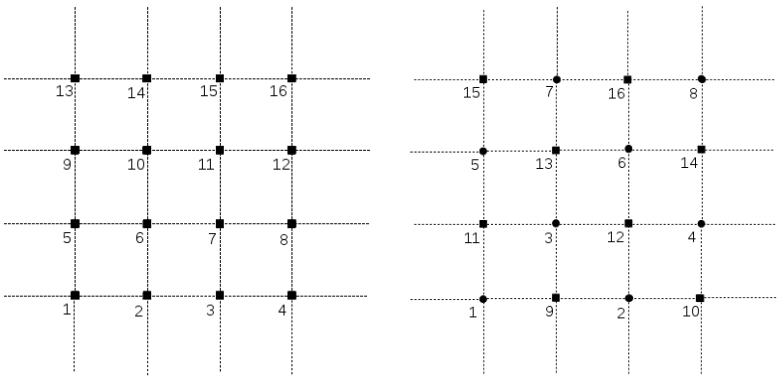
   $$\Omega_k^{black} = \{(ih_k, jh_k) \in \Omega_k : i + j \text{ odd}\}.$$

   The red points are enumerated in their lexicographical ordering first, then the black points are ordered in a similar way.

3. *Four-color ordering*:
   Divide Ω_k into four subsets

   $$\begin{aligned}
   \Omega_k^1 &= \{(ih_k, jh_k) \in \Omega_k : i, j \text{ even}\},\\
   \Omega_k^2 &= \{(ih_k, jh_k) \in \Omega_k : i, j \text{ odd}\},\\
   \Omega_k^3 &= \{(ih_k, jh_k) \in \Omega_k : i \text{ even}, j \text{ odd}\},\\
   \Omega_k^4 &= \{(ih_k, jh_k) \in \Omega_k : i \text{ odd}, j \text{ even}\}.
   \end{aligned}$$

   The grid points are numbered in turn Ω_k^1, Ω_k^2, Ω_k^3, Ω_k^4 and in each subset Ω_k^i the points are ordered lexicographically.



(a) lexicographic ordering (b) red-black ordering

Figure 3.3: Ordering

The checkerboard ordering is well-suited for five-point discretizations, whereas the four-color ordering can be used for general nine-point schemes. If the red-black ordering is used, another variation of the Gauss-Seidel iteration is motivated: the *red-black Gauss-Seidel (GS-RB)*. Every iteration consists of two half-steps. In the first half-step, all black points are treated simultaneously and independently and in the second half-step, all red points are treated, using the already updated values in the black points. Each half-step is effectively a (JAC) iteration for the black resp. red grid points. This considerations result in the following iteration scheme:

First half-step

$$u_k^{(l+\frac{1}{2})} = \begin{cases} -(D_k + L_k)^{-1} R_k u_k^{(l)} + (D_k + L_k)^{-1} f_k & \text{if } u_k \in \Omega_k^{red} \\ u_k^{(l)} & \text{else} \end{cases}$$

and the second half-step

$$u_k^{(l+1)} = \begin{cases} -(D_k + L_k)^{-1} R_k u_k^{(l+\frac{1}{2})} + (D_k + L_k)^{-1} f_k & \text{if } u_k \in \Omega_k^{black} \\ u_k^{(l+\frac{1}{2})} & \text{else.} \end{cases}$$

3.3.2.3 Blockwise Iterations

An extension of the (pointwise) iteration schemes is to solve the linear system with respect to blocks of unknowns. Let $(u_{k,1}, u_{k,2}, \dots)$ be an ordering of the unknowns and the index set $I_k = \{1, \dots, n_k\}$ be partitioned into $p = p_k$ subsets

$$I_k^j = \{i_{j-1} + 1, i_{j-1} + 2, \dots, i_j\} \quad \text{for } 1 \le j \le p$$

with $0 = i_0 < i_1 < \cdots < i_p = n_k$. The unknowns $u_k^j := (u_{k,i})_{i \in I_k^j}$ form the j-th block of u_k. The matrix $A_k = D_k + L_k + R_k$ is splitted, where D_k is blockdiagonal and L_k and R_k are strictly lower respectively upper block-triangular. A matrix L_k is strictly lower block-triangular if $L_{k,\nu\mu} = 0$ whenever $\nu \in I_k^j, \mu \in I_k^{j'}$ with $j' \ge j$ and analogously a matrix R_k is strictly upper block-triangular if $L_{k,\nu\mu} = 0$ whenever $\nu \in I_k^j, \mu \in I_k^{j'}$ with $j' \le j$.

- *Block Jacobi iteration*:
  The Block Jacobi iteration is given by

$$\begin{aligned} u_k^{(l+1)} &= D_k^{-1}(D_k - A_k)u_k^{(l)} + D_k^{-1}f_k \\ \Leftrightarrow u_k^{(l+1)} &= u_k^{(l)} + D_k^{-1}\left(f_k - A_k u_k^{(l)}\right). \end{aligned}$$

  This iteration is independent of the ordering of the blocks and the ordering of the unknowns within the blocks.

- *Block Gauss-Seidel iteration*:

$$u_k^{(l+1)} = -(D_k + L_k)^{-1}R_k u_k^{(l)} + (D_k + L_k)^{-1}f_k$$

  and written in componentwise notation omitting the level index k

$$u_i^{(l+1)} = A_{ii}^{-1}\left(f_i - \sum_{j=1}^{i-1} A_{ij}u_j^{(l+1)} - \sum_{j=i+1}^{n} A_{ij}u_j^{(l)}\right) \quad \text{for } i = 1, \ldots, p.$$

  The blockwise Gauss-Seidel iteration depends on the ordering of the blocks, the numbering within the blocks is of no account.

Analogously to the scalar red-black ordering, the blocks can be separated into red and black blocks and then treated analogously to the pointwise GS-RB variation.

3.3.2.4 Parallel Properties of Smoothers

Comparing the parallelization capability of the smoothers, the (damped) Jacobi-iteration is fully parallelizable as the operation can be applied to all grid points Ω_h simultaneously. The new values do not depend on each other. The degree of parallelism defined by [53, Chapter 2.1.4] as the number of grid points that can be treated simultaneously by the smoothing operator, is

$$\text{par-deg}(\omega\text{-JAC}) = \#\Omega_k.$$

Within the Gauss-Seidel iteration there are dependencies between the grid point as one wants to use the most recent values of u_k wherever possible for the update. The grid points on the diagonals in Ω_k are independent of each other for the five-point discretization and can be treated in parallel. As the number of grid points on the diagonal varies, the degree of parallelism is bounded by

$$\text{par-deg}(\text{GS-LEX}) \leq \sqrt{\#\Omega_k}.$$

The red-black Gauss-Seidel iteration consists of two half-steps, which are both fully parallelizable. Here the degree of parallelism is

$$\text{par-deg}(\text{GS-RB}) = \frac{1}{2}\#\Omega_k.$$

3.3.3 Coarse Grid Correction

To solve the defect equation approximately, the idea is to use an appropriate approximation A_{k-1} of A_k on a coarser grid Ω_{k-1}. Therefore a hierarchy of grids Ω_k has to be determined, as illustrated in Figure 3.4. The most convenient choice, called *standard coarsening*, defines a sequence of grid sizes $\{h_k\}_{k=0}^{\infty}$, where $h_0 = \frac{1}{2}$ and $h_k = \frac{h_0}{2^k}$ for $k > 0$. The grid Ω_0 consists of only one interior grid point. The choice of the grid hierarchy Ω_k defines a family of linear systems $A_k u_k = f_k$.

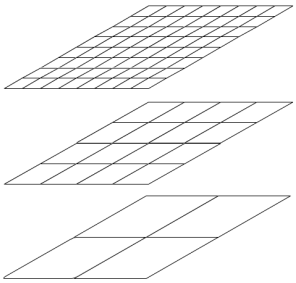


Figure 3.4: Standard coarsening

The coarse grid correction consists of:

Algorithm 3.2 Coarse grid correction

1: Compute the defect: $d_k^{(l)} = f_k - A_k u_k^{(l)}$.
2: Restriction: $d_{k-1}^{(l)} = I_k^{k-1} d_k^{(l)}$.
3: Solve on Ω_{k-1}: $A_{k-1} e_{k-1}^{(l)} = d_{k-1}^{(l)}$.
4: Prolongation: $e_k^{(l)} = I_{k-1}^{k} e_{k-1}^{(l)}$.
5: Update approximation: $u_k^{(l+1)} = u_k^{(l)} + e_k^{(l)}$.

REMARK 3.23.
Taken on its own, the coarse grid correction procedure is not convergent as

$$\rho\left(I_k - I_{k-1}^{k}\left(A_{k-1}^{-1} I_k^{k-1} A_k\right)\right) > 1,$$

see [40, Lemma 4.41].

In this section, the miscellaneous components of the CGC are considered.

3.3.3.1 Coarse Grid Operator

Once reached a predetermined coarsest grid Ω_p, $0 \leq p \leq L$ there are basically two possibilities for solving the defect equation $A_p e_p = d_p$: On the one hand, a direct solver and on the other hand an iterative solver. A direct solver can be applied if the coarsest grid is sufficiently small - in the extreme case $p = 0$ there is only one scalar equation left and can be solved analytically. If the coarsest grid is finer, an iterative solver, as the smoothing iterations introduced in the last section or Krylov-subspace methods, can be applied to solve the defect equation up to a desired accuracy.

3.3.3.2 Transfer Operator: Prolongation

The *prolongation (interpolation) operator* I_{k-1}^{k} is a linear and injective mapping which maps a grid function $u_{k-1} \in U_{k-1}$ to a grid function $u_k \in U_k$. In two dimensions, for a given grid point $(x_1, x_2) = (ih_k, jh_k) \in \Omega_k$ a piecewise *bilinear interpolation*

$$
\begin{aligned}
u_k(x_1, x_2) &= I_{k-1}^{k} u_{k-1}(x_1, x_2) \\
&= \begin{cases}
u_{k-1}(x_1, x_2) & \frac{x_1}{h_k}, \frac{x_2}{h_k} \text{ even} \\
\frac{1}{2}\left[u_{k-1}(x_1, x_2 + h_k) + u_{k-1}(x_1, x_2 - h_k)\right] & \frac{x_1}{h_k} \text{ even}, \frac{x_2}{h_k} \text{ odd} \\
\frac{1}{2}\left[u_{k-1}(x_1 + h_k, x_2) + u_{k-1}(x_1 - h_k, x_2)\right] & \frac{x_1}{h_k} \text{ odd}, \frac{x_2}{h_k} \text{ even} \\
\frac{1}{4}[u_{k-1}(x_1 + h_k, x_2 + h_k) + u_{k-1}(x_1 + h_k, x_2 - h_k) & \\
\quad + u_{k-1}(x_1 - h_k, x_2 + h_k) + u_{k-1}(x_1 - h_k, x_2 - h_k)] & \frac{x_1}{h_k}, \frac{x_2}{h_k} \text{ odd}
\end{cases}
\end{aligned}
$$

is applied, see Figure 3.5. As abbreviation, the stencil notation is used:

$$
I_{k-1}^{k} = \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_{2h_k}^{h_k}.
$$

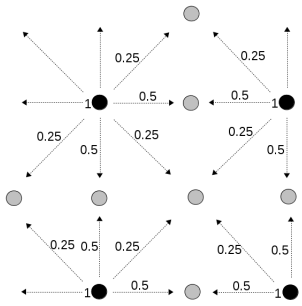


Figure 3.5: Prolongation in two dimensions

This interpolation is also called *nine-point prolongation*. There are even more other possibilities [29, Chapter 3.4].
For three dimensions, the *trilinear interpolation* in stencil notation is given by

$$
I_{k-1}^{k} = \frac{1}{8} \left[\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 4 & 8 & 4 \\ 2 & 4 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right]_{2h_k}^{h_k},
$$

defined by

$$\begin{aligned} u_k(x_1,x_2,x_3) &= I_{k-1}^k u_k(x_1,x_2,x_3) \\ &= \frac{1}{8}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}_{2h_k}^{h_k} u_{k-1}(x_1,x_2,x_3-h_k) \\ &+ \frac{1}{8}\begin{bmatrix}2&4&2\\4&8&4\\2&4&2\end{bmatrix}_{2h_k}^{h_k} u_{k-1}(x_1,x_2,x_3) \\ &+ \frac{1}{8}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}_{2h_k}^{h_k} u_{k-1}(x_1,x_2,x_3+h_k). \end{aligned}$$

3.3.3.3 Transfer Operator: Restriction

The *restriction operator* I_k^{k-1} is a linear and surjective mapping that maps a grid function $u_k \in U_k$ to a grid function $u_{k-1} \in U_{k-1}$. The easiest way is the *trivial restriction* defined by

$$u_k(x_1,x_2) = u_{k-1}(x_1,x_2) \quad \text{for } (x_1,x_2) \in \Omega_{k-1}.$$

There are several disadvantages, in the first place the loss of information [29, Chapter 3.5]. A common choice and frequently used is the *full weighting operator*, a nine-point weighted average (*nine-point restriction*) of u_k, in stencil notation for the two-dimensional case

$$I_k^{k-1} = \frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}_{h_k}^{2h_k}$$

and in the three-dimensional case a 27 point weighted average

$$I_k^{k-1} = \frac{1}{64}\left[\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}\begin{bmatrix}2&4&2\\4&8&4\\2&4&2\end{bmatrix}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}\right]_{h_k}^{2h_k},$$

Applying this operator to a grid function $u_k(x_1,x_2)$ at a coarse grid point $(x_1,x_2) \in \Omega_{k-1}$ is illustrated in Figure 3.6 and formulated by

$$\begin{aligned} u_{k-1}(x_1,x_2) =& I_k^{k-1} u_k(x_1,x_2) \\ =& \frac{1}{16}[4u_k(x_1,x_2) + 2u_k(x_1+h_k,x_2) + 2u_k(x_1-h_k,x_2) + 2u_k(x_1,x_2+h_k) \\ &+ 2u_k(x_1,x_2-h_k) + u_k(x_1+h_k,x_2+h_k) + u_k(x_1+h_k,x_2-h_k) \\ &+ u_k(x_1-h_k,x_2+h_k) + u_k(x_1-h_k,x_2-h_k)]. \end{aligned}$$

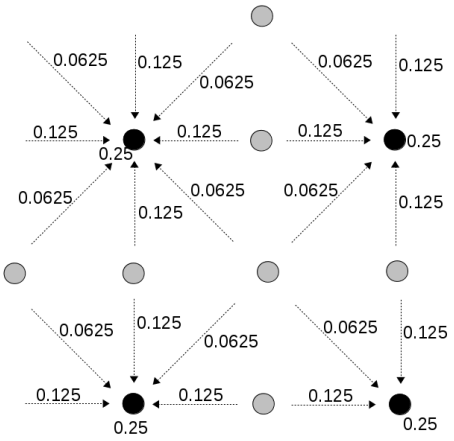


Figure 3.6: Restriction in two dimensions

For the three-dimensional case, it holds

$$
\begin{aligned}
u_{k-1}(x_1,x_2,x_3) &= I_k^{k-1}u_k(x_1,x_2,x_3)\\
&=\frac{1}{64}[8u_k(x_1,x_2,x_3)+4u_k(x_1+h_k,x_2,x_3)+4u_k(x_1-h_k,x_2,x_3)\\
&\quad+4u_k(x_1,x_2+h_k,x_3)+4u_k(x_1,x_2-h_k,x_3)+4u_k(x_1,x_2,x_3+h_k)\\
&\quad+4u_k(x_1,x_2,x_3-h_k)+2u_k(x_1+h_k,x_2+h_k,x_3)+2u_k(x_1+h_k,x_2,x_3+h_k)\\
&\quad+2u_k(x_1,x_2+h_k,x_3+h_k)+2u_k(x_1+h_k,x_2-h_k,x_3)+2u_k(x_1+h_k,x_2,x_3-h_k)\\
&\quad+2u_k(x_1,x_2+h_k,x_3-h_k)+2u_k(x_1-h_k,x_2+h_k,x_3)+2u_k(x_1-h_k,x_2,x_3+h_k)\\
&\quad+2u_k(x_1,x_2-h_k,x_3+h_k)+2u_k(x_1-h_k,x_2-h_k,x_3)+2u_k(x_1-h_k,x_2,x_3-h_k)\\
&\quad+2u_k(x_1,x_2-h_k,x_3-h_k)+u_k(x_1+h_k,x_2+h_k,x_3+h_k)+u_k(x_1+h_k,x_2+h_k,x_3-h_k)\\
&\quad+u_k(x_1+h_k,x_2-h_k,x_3+h_k)+u_k(x_1+h_k,x_2-h_k,x_3-h_k)\\
&\quad+u_k(x_1-h_k,x_2+h_k,x_3+h_k)+u_k(x_1-h_k,x_2+h_k,x_3-h_k)\\
&\quad+u_k(x_1-h_k,x_2-h_k,x_3+h_k)+u_k(x_1-h_k,x_2-h_k,x_3-h_k)].
\end{aligned}
$$

There are many other possibilities for choosing the restriction operator, see e.g. [53, Chapter 2.3.3].

REMARK 3.24.
The nine-point restriction I_k^{k-1} is the adjoint to the nine-point prolongation I_{k-1}^k in the following sense

$$\left(I_k^{k-1}u_k, v_{k-1}\right)_{k-1} = \left(u_k, I_{k-1}^k v_{k-1}\right)_k \quad \forall u_k \in U_k, v_{k-1} \in U_{k-1},$$

see Section 3.4.3.2.

3.4 Multigrid Solver for the Optimality System

In this section the *collective smoothing multigrid approach (CSMG)* proposed by Borzì [7, Chapter 5.7], [5] and [6] is introduced and investigated. Originally multigrid methods are used

for solving linear systems arising from the discretization of partial differential equations. For the special case of the optimal control problem with tracking-type objective and self-adjoint PDE-constraint, a solution of the optimality system within the multigrid framework is possible. For investigating the convergence properties of the multigrid method applied to the optimality system, two complementary analytic frameworks are used. On the one hand, the two-grid local Fourier analysis in Section 3.4.2 and on the other hand the general multigrid convergence theory for the special case of nonsymmetric systems in Section 3.4.3.

3.4.1 Collective Smoothing Multigrid Approach

For generalizing the multigrid method and in particular the scalar smoothing scheme to systems of equations, like the optimality system, the most natural extension is to replace the relaxation smoothers by collective relaxation smoothers. In this thesis the CSMG approach is considered, which means to solve the optimality system arising from optimal control problems for the state, the adjoint and the control variables simultaneously in the multigrid process by using collective smoothers for the optimizations variables. A survey of the development and the field of application of this approach can be found in [7, Chapter 5.7] and the references given therein. In this section, the CSMG scheme for the academic elliptic distributed control problem introduced in Section 2.5

$$\min_{(y,u)} J(y,u) := \min_{(y,u)} \frac{1}{2}\|y - y_d\|^2_{L^2(\Omega)} + \frac{\alpha}{2}\|u\|^2_{L^2(\Omega)}$$

$$\begin{aligned} Ly &= f + u \quad \text{in } \Omega \\ y &= 0 \qquad \text{on } \partial\Omega \end{aligned}$$

and the resulting optimality system after eliminating the control u for $\alpha > 0$

$$\begin{aligned} \alpha Ly - p &= \alpha f \quad \text{in } \Omega \\ y &= 0 \qquad \text{on } \partial\Omega \\ y + Lp &= y_d \quad \text{in } \Omega \\ p &= 0 \qquad \text{on } \partial\Omega \end{aligned}$$

is presented. For the treatment of control constraints see [5].
To avoid technical difficulties consider the constant coefficient case in two dimensions, i.e. $a_{11}(x_1, x_2) \equiv a_{11}$, $a_{22}(x_1, x_2) \equiv a_{22}$, $c(x_1, x_2) \equiv c$ and $\Omega = (0,1)^2$ the unit square. The resulting five-point discretization of the differential operator L is given by

$$L_h = -a_{11} D^-_{x_1} D^+_{x_1} - a_{22} D^-_{x_2} D^+_{x_2} + c$$

and written in stencil notation

$$\frac{1}{h^2} \begin{bmatrix} 0 & -a_{22} & 0 \\ -a_{11} & 2(a_{11}+a_{22}) & -a_{11} \\ 0 & -a_{22} & 0 \end{bmatrix}_h + \begin{bmatrix} 0 & 0 & 0 \\ 0 & c & 0 \\ 0 & 0 & 0 \end{bmatrix}_h .$$

The resulting discrete operator

$$\mathcal{A}_h = \begin{bmatrix} \alpha L_h & -I_h \\ I_h & L_h \end{bmatrix}$$

was investigated in Section 3.2.1. Let Ω_h be a uniform cartesian grid on Ω, i.e. for $i, j = 1, \ldots, n$, denoting n the number of internal grid points in x_1-direction, the grid points $(x_i, x_j) \in \Omega_h$ are defined by $x_i = ih, x_j = jh$ and $h = \frac{1}{n+1}$. Both of the differential operators L resp. αL are discretized with same finite difference approximation. The idea of the CSMG is to consider one grid for both differential operators and treat the variables at each grid point collectively. For a given grid point, the discrete optimality system with a lexicographic ordering is given by

$$\begin{aligned}
&\frac{\alpha}{h^2}[(2(a_{11}+a_{22})+h^2c)y(x_1,x_2) - a_{11}y(x_1-h,x_2) - a_{11}y(x_1+h,x_2) \\
&\qquad -a_{22}y(x_1,x_2-h) - a_{22}y(x_1,x_2+h)] - p(x_1,x_2) = \alpha f(x_1,x_2) \\
&\frac{1}{h^2}[(2(a_{11}+a_{22})+h^2c)p(x_1,x_2) - a_{11}p(x_1-h,x_2) - a_{11}p(x_1+h,x_2) \\
&\qquad -a_{22}p(x_1,x_2-h) - a_{22}p(x_1,x_2+h)] + y(x_1,x_2) = y_d(x_1,x_2)
\end{aligned}$$

and collectively in point-block notation

$$\begin{aligned}
&\begin{bmatrix} \frac{\alpha}{h^2}(2(a_{11}+a_{22})+h^2c) & -1 \\ 1 & \frac{1}{h^2}(2(a_{11}+a_{22})+h^2c) \end{bmatrix} \begin{pmatrix} y(x_1,x_2) \\ p(x_1,x_2) \end{pmatrix} \\
&- \begin{bmatrix} \frac{\alpha}{h^2}a_{11} & 0 \\ 0 & \frac{1}{h^2}a_{11} \end{bmatrix} \begin{pmatrix} y(x_1-h,x_2) \\ p(x_1-h,x_2) \end{pmatrix} - \begin{bmatrix} \frac{\alpha}{h^2}a_{11} & 0 \\ 0 & \frac{1}{h^2}a_{11} \end{bmatrix} \begin{pmatrix} y(x_1+h,x_2) \\ p(x_1+h,x_2) \end{pmatrix} \\
&- \begin{bmatrix} \frac{\alpha}{h^2}a_{22} & 0 \\ 0 & \frac{1}{h^2}a_{22} \end{bmatrix} \begin{pmatrix} y(x_1,x_2+h) \\ p(x_1,x_2+h) \end{pmatrix} - \begin{bmatrix} \frac{\alpha}{h^2}a_{22} & 0 \\ 0 & \frac{1}{h^2}a_{22} \end{bmatrix} \begin{pmatrix} y(x_1,x_2-h) \\ p(x_1,x_2-h) \end{pmatrix} \\
&= \begin{pmatrix} \alpha f(x_1,x_2) \\ y_d(x_1,x_2) \end{pmatrix}.
\end{aligned}$$

Smoothing

A general collective smoothing step at $(x_1, x_2) \in \Omega_k$ consists of updating the values $y(x_1, x_2)$ and $p(x_1, x_2)$ such that the resulting residuals of the two equations at that point are zero. The neighboring variables are considered as constant ones. This method is originally developed for the control constrained case in [7, Chapter 5.7].

$$\begin{pmatrix} y(x_1,x_2) \\ p(x_1,x_2) \end{pmatrix} = \begin{bmatrix} \frac{\alpha}{h^2}(2(a_{11}+a_{22})+h^2c) & -1 \\ 1 & \frac{1}{h^2}(2(a_{11}+a_{22})+h^2c) \end{bmatrix}^{-1} \Bigg[\begin{pmatrix} \alpha f(x_1,x_2) \\ y_d(x_1,x_2) \end{pmatrix} \tag{3.8}$$

$$+ \begin{bmatrix} \frac{\alpha}{h^2}a_{11} & 0 \\ 0 & \frac{1}{h^2}a_{11} \end{bmatrix} \begin{pmatrix} y(x_1+h,x_2) \\ p(x_1+h,x_2) \end{pmatrix} + \begin{bmatrix} \frac{\alpha}{h^2}a_{11} & 0 \\ 0 & \frac{1}{h^2}a_{11} \end{bmatrix} \begin{pmatrix} y(x_1-h,x_2) \\ p(x_1-h,x_2) \end{pmatrix} \tag{3.9}$$

$$+ \begin{bmatrix} \frac{\alpha}{h^2}a_{22} & 0 \\ 0 & \frac{1}{h^2}a_{22} \end{bmatrix} \begin{pmatrix} y(x_1,x_2+h) \\ p(x_1,x_2+h) \end{pmatrix} + \begin{bmatrix} \frac{\alpha}{h^2}a_{22} & 0 \\ 0 & \frac{1}{h^2}a_{22} \end{bmatrix} \begin{pmatrix} y(x_1,x_2-h) \\ p(x_1,x_2-h) \end{pmatrix} \Bigg]. \tag{3.10}$$

Treating the linear system in this point-block structure, it is obvious to consider block iterations as smoother, like the block Jacobi or the red-black ordered block Gauss-Seidel iteration where the blocks are separated in red and black blocks. For this iteration, one smoothing step is defined by two half-steps.

First half-step: red blocks

$$\begin{aligned}
u^{(l+\frac{1}{2})}(x_1,x_2) &= \begin{pmatrix} y(x_1,x_2) \\ p(x_1,x_2) \end{pmatrix}^{(l+\frac{1}{2})} \\
&= D^{-1}\Bigg[\begin{pmatrix} \alpha f(x_1,x_2) \\ y_d(x_1,x_2) \end{pmatrix} - \Bigg(\begin{bmatrix} \frac{\alpha}{h^2}a_{11} & 0 \\ 0 & \frac{1}{h^2}a_{11} \end{bmatrix} \begin{pmatrix} y(x_1-h,x_2) \\ p(x_1-h,x_2) \end{pmatrix}^{(l)} \\
&+ \begin{bmatrix} \frac{\alpha}{h^2}a_{11} & 0 \\ 0 & \frac{1}{h^2}a_{11} \end{bmatrix} \begin{pmatrix} y(x_1+h,x_2) \\ p(x_1+h,x_2) \end{pmatrix}^{(l)} + \begin{bmatrix} \frac{\alpha}{h^2}a_{22} & 0 \\ 0 & \frac{1}{h^2}a_{22} \end{bmatrix} \begin{pmatrix} y(x_1,x_2-h) \\ p(x_1,x_2-h) \end{pmatrix}^{(l)} \\
&+ \begin{bmatrix} \frac{\alpha}{h^2}a_{22} & 0 \\ 0 & \frac{1}{h^2}a_{22} \end{bmatrix} \begin{pmatrix} y(x_1,x_2+h) \\ p(x_1,x_2+h) \end{pmatrix}^{(l)} \Bigg) \Bigg]
\end{aligned}$$

and the second half step: black blocks

$$\begin{aligned}
u^{(l+1)}(x_1,x_2) &= \begin{pmatrix} y(x_1,x_2) \\ p(x_1,x_2) \end{pmatrix}^{(l+1)} \\
&= D^{-1}\Bigg[\begin{pmatrix} \alpha f(x_1,x_2) \\ y_d(x_1,x_2) \end{pmatrix} - \Bigg(\begin{bmatrix} \frac{\alpha}{h^2}a_{11} & 0 \\ 0 & \frac{1}{h^2}a_{11} \end{bmatrix} \begin{pmatrix} y(x_1-h,x_2) \\ p(x_1-h,x_2) \end{pmatrix}^{(l+\frac{1}{2})} \\
&+ \begin{bmatrix} \frac{\alpha}{h^2}a_{11} & 0 \\ 0 & \frac{1}{h^2}a_{11} \end{bmatrix} \begin{pmatrix} y(x_1+h,x_2) \\ p(x_1+h,x_2) \end{pmatrix}^{(l+\frac{1}{2})} + \begin{bmatrix} \frac{\alpha}{h^2}a_{22} & 0 \\ 0 & \frac{1}{h^2}a_{22} \end{bmatrix} \begin{pmatrix} y(x_1,x_2-h) \\ p(x_1,x_2-h) \end{pmatrix}^{(l+\frac{1}{2})} \\
&+ \begin{bmatrix} \frac{\alpha}{h^2}a_{22} & 0 \\ 0 & \frac{1}{h^2}a_{22} \end{bmatrix} \begin{pmatrix} y(x_1,x_2+h) \\ p(x_1,x_2+h) \end{pmatrix}^{(l+\frac{1}{2})} \Bigg) \Bigg].
\end{aligned}$$

The inverse D^{-1} of

$$D = \begin{bmatrix} \frac{\alpha}{h^2}(2(a_{11}+a_{22})+h^2c) & -1 \\ 1 & \frac{1}{h^2}(2(a_{11}+a_{22})+h^2c) \end{bmatrix}$$

can be determined analytically

$$D^{-1} = \frac{1}{\frac{\alpha}{h^4}(2(a_{11}+a_{22})+h^2c)^2+1} \begin{bmatrix} \frac{1}{h^2}(2(a_{11}+a_{22}+h^2c)) & 1 \\ -1 & \frac{\alpha}{h^2}(2(a_{11}+a_{22})+h^2c) \end{bmatrix}.$$

Solver coarsest grid
On the coarsest grid Ω_0, i.e. $h_0 = \frac{1}{2}$, the 2×2 system can be solved exactly by

$$\begin{pmatrix} y(\frac{1}{2},\frac{1}{2}) \\ p(\frac{1}{2},\frac{1}{2}) \end{pmatrix} = \frac{1}{16\alpha(2(a_{11}+a_{22})+\frac{c}{4})^2+1} \begin{bmatrix} 4(2(a_{11}+a_{22})+\frac{c}{4}) & 1 \\ -1 & 4\alpha(2(a_{11}+a_{22})+\frac{c}{4}) \end{bmatrix} \begin{pmatrix} \alpha f(\frac{1}{2},\frac{1}{2}) \\ y_d(\frac{1}{2},\frac{1}{2}) \end{pmatrix}.$$

For the special case $L = -\Delta$, this simplifies to

$$\begin{pmatrix} y(\frac{1}{2},\frac{1}{2}) \\ p(\frac{1}{2},\frac{1}{2}) \end{pmatrix} = \frac{1}{256\alpha+1} \begin{bmatrix} 16 & 1 \\ -1 & 16\alpha \end{bmatrix} \begin{pmatrix} \alpha f(\frac{1}{2},\frac{1}{2}) \\ y_d(\frac{1}{2},\frac{1}{2}) \end{pmatrix}.$$

3.4.2 Local Fourier Analysis

The local Fourier analysis was introduced by Brandt [14] and extended and refined in several papers, e.g. [15]. The purpose of this analysis is to obtain sharp convergence estimates by simplifying assumptions on the boundary conditions. Following the formalism of [53, Chapter 4], the main idea is the local nature of the local Fourier analysis:
Under general assumptions, any general discrete operator, possibly nonlinear, possibly with nonconstant coefficients, can be linearized locally and can be replaced locally by an operator with constant coefficients. With this simplification, general linear discrete operators with constant coefficients are considered in local Fourier analysis. Formally, they are defined on an infinite grid. This fact seems to be deficient as the influence of the boundaries and boundary conditions are neglected, but the objective of the local Fourier analysis is to determine the quantitative convergence behavior and efficiency of an appropriate multigrid algorithm can attain if a proper boundary treatment is included.

3.4.2.1 Theoretical Background

Firstly a brief introduction in terminology and notation is given. To avoid technical complications, considering the discretization of the constant case uniformly elliptic operator

$$L = -\frac{\partial^2}{\partial x_1^2} - \frac{\partial^2}{\partial x_2^2} = -\Delta \tag{3.11}$$

in two dimensions. Later, this is generalized for the coupled system of PDEs. For the multigrid solver, standard coarsening, full-weighting and bilinear interpolation are assumed. The local Fourier analysis is of local nature: the aim is to determine quantities for the smoothing iteration $\mu_{loc}(S_h)$ and two-grid convergence factors $\rho_{loc}(TG_h^H)$ for the insight into the asymptotic convergence behavior. For the local Fourier analysis define an infinite grid G_h on $\mathbb{R}^2$ and a special type of grid functions.

DEFINITION 3.25 (Grid, grid function)**.**
For a fixed mesh size $h = (h_1, h_2)$ define

1. infinite grid
$$G_h = \{x \in \mathbb{R}^2 : x = jh := (j_1 h_1, j_2 h_2), j = (j_1, j_2) \in \mathbb{Z}^2\}$$
2. grid function
$$\varphi_h(\theta, x) = e^{i\theta \cdot x/h} := e^{i\theta_1 x_1/h_1} e^{i\theta_2 x_2/h_2} \quad \text{for } x \in G_h.$$

REMARK 3.26.
For continuously varying θ in $\mathbb{R}^2$, it is sufficient to consider

$$\varphi(\theta, x) \quad \text{with } \theta \in [-\pi, \pi)^2$$

as

$$\varphi(\theta, x) \equiv \varphi(\theta', x) \quad \text{for } x \in G_h$$

if and only if

$$\theta_1 = \theta_1' (\text{mod } 2\pi) \text{ and } \theta_2 = \theta_2' (\text{mod } 2\pi).$$

These grid functions are linearly independent on G_h.

On an infinite grid G_h, discrete differential operators L_h are considered corresponding to a difference stencil.

DEFINITION 3.27 (Difference stencil)**.**
Let u_h be a grid function on G_h, $h = (h_1, h_2)$. A general difference stencil

$$[s_{\kappa_1\kappa_2}]_h = \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \cdots & s_{-1,1} & s_{0,1} & s_{1,1} & \cdots \\ \cdots & s_{-1,0} & s_{0,0} & s_{1,0} & \cdots \\ \cdots & s_{-1,-1} & s_{0,-1} & s_{1,-1} & \cdots \\ & \vdots & \vdots & \vdots & \end{bmatrix}_h$$

defines an operator on the set of grid functions by

$$[s_{\kappa_1\kappa_2}]_h u_h(x_1, x_2) = \sum_{(\kappa_1,\kappa_2)} s_{\kappa_1\kappa_2} u_h(x_1 + \kappa_1 h_1, x_2 + \kappa_2 h_2).$$

Assume that only a finite number of coefficients $s_{\kappa_1\kappa_2}$ are nonzero. A typical example for a stencil is the already mentioned five point stencil

$$\begin{bmatrix} & s_{0,1} & \\ s_{-1,0} & s_{0,0} & s_{1,0} \\ & s_{0,-1} & \end{bmatrix}_h$$

and for the constant coefficient operator (3.11) $-\Delta_h$

$$-\frac{1}{h^2}\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}_h .$$

The discrete operators can be defined by its difference stencil

$$L_h \hat{=} [s_\kappa]_h \quad (\kappa = (\kappa_1, \kappa_2) \in \mathbb{Z}^2)$$

and applied to a grid function u_h

$$L_h u_h(x) = \sum_{\kappa \in V} s_\kappa u_h(x + \kappa h)$$

with constant coefficients $s_\kappa \in \mathbb{R}$ and V being a finite index set.

LEMMA 3.28.
For $\theta \in [-\pi, \pi)^2$, all grid functions $\varphi(\theta, x)$ are (formal) eigenfunctions of any discrete operator which can be described by a difference stencil $[s_\kappa]_h$. The relation

$$L_h \varphi(\theta, x) = \tilde{L}_h(\theta)\varphi(\theta, x) \quad (x \in G_h)$$

holds with

$$\tilde{L}_h(\theta) = \sum_\kappa s_\kappa e^{i\theta\cdot\kappa}.$$

$\tilde{L}_h(\theta)$ is called *formal eigenvalue* or the *symbol of* L_h.

Proof. [53, Lemma 4.2.1]. □

EXAMPLE 3.29 (Laplace operator)**.**
The symbol for the discrete Laplace operator $L_h = -\Delta_h$ is

$$\begin{aligned}\tilde{L}_h(\theta) &= \frac{1}{h^2}\left(4 - \left(e^{i\theta_1} + e^{i\theta_2} + e^{-i\theta_1} + e^{-i\theta_2}\right)\right) \\ &= \frac{2}{h^2}\left(2 - (\cos\theta_1 + \cos\theta_2)\right).\end{aligned}$$

For the investigation of convergence estimates for the coarse grid correction, a coarse grid is defined.

DEFINITION 3.30 (Coarse grid)**.**
Let G_h an infinite grid. A coarse grid

$$G_H = \{x = \kappa H : \kappa \in \mathbb{Z}^2\}$$

is obtained by standard coarsening of G_h, i.e. $H = (2h_1, 2h_2)$.

Only those frequency components

$$\varphi(\theta, \cdot) \quad \text{with } -\frac{\pi}{2} \leq \theta < \frac{\pi}{2}$$

are distinguishable on G_H, i.e.

$$\varphi(\theta, x) = \varphi(\theta', x) \text{ for } x \in G_H \Leftrightarrow \theta = \theta' (\text{mod } \pi).$$

This leads to the distinction of high and low frequencies.

DEFINITION 3.31 (High and low frequency components)**.**

$$\begin{aligned}\varphi \text{ low frequency component } &\Leftrightarrow \theta \in T^{low} := \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)^2 \\ \varphi \text{ high frequency component } &\Leftrightarrow \theta \in T^{high} := [-\pi, \pi)^2 \setminus \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)^2\end{aligned}$$

3.4.2.2 Smoothing Analysis

One goal of the local Fourier analysis is to determine smoothing factors for S_h. Consider the optimality system

$$\begin{bmatrix} -\alpha\Delta_h & -I_h \\ I_h & -\Delta_h \end{bmatrix} \begin{pmatrix} y_h \\ p_h \end{pmatrix} = \begin{pmatrix} \alpha f_h \\ y_d \end{pmatrix} (\Leftrightarrow: \mathcal{A}_h w_h = \phi_h)$$

and assume that the smoothing iteration can locally be written as

$$\mathcal{A}_h^+ \bar{w}_h + \mathcal{A}_h^- w_h = \phi_h$$

where w_h corresponds to the 'old' approximation (before the smoothing step) of w_h. In this sense, the relaxation is characterized by a splitting of the form

$$\mathcal{A}_h = \mathcal{A}_h^+ + \mathcal{A}_h^-.$$

For the analysis of different smoothers, the notation of [5] is recapitulated: define

$$\mathcal{B}_h^+ = \begin{bmatrix} \alpha \sum_h^+ & 0 \\ 0 & \sum_h^+ \end{bmatrix}, \mathcal{B}_h^- = \begin{bmatrix} \alpha \sum_h^- & 0 \\ 0 & \sum_h^- \end{bmatrix}, \mathcal{D}_h^+ = \begin{bmatrix} \frac{4\alpha}{h^2} I_h & -I_h \\ I_h & \frac{4}{h^2} I_h \end{bmatrix},$$

where $\sum_h^-, \sum_h^+$ are given in stencil form:

$$\sum_h^+ = \tfrac{1}{h^2} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \sum_h^- = \tfrac{1}{h^2} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

For GS-LEX, $\mathcal{A}_h$ is splitted into

$$\begin{aligned} \mathcal{A}_h^+ &= \mathcal{D}_h - \mathcal{B}_h^+ \\ \mathcal{A}_h^- &= -\mathcal{B}_h^-. \end{aligned}$$

Let $\tilde{\mathcal{A}}_h^+$ and $\tilde{\mathcal{A}}_h^-$ be the symbols of $\mathcal{A}_h^+$ and $\mathcal{A}_h^-$.

LEMMA 3.32.
Under the splitting assumption, all $\varphi(\theta, \cdot)$ with $\tilde{\mathcal{A}}_h^+(\theta) \neq 0$ are eigenfunctions of S_h:

$$S_h \varphi(\theta, x) = \tilde{S}_h(\theta) \varphi(\theta, x) \quad (-\pi \leq \theta \leq \pi)$$

with the amplification factor

$$\tilde{S}_h(\theta) := -\tilde{\mathcal{A}}_h^+(\theta)^{-1} \tilde{\mathcal{A}}_h^-(\theta).$$

Proof. Straightforward extension of [53, Lemma 4.3.1] for general systems of PDEs. □

For GS-LEX, the symbols are given by

$$\tilde{\mathcal{A}}_h^+(\theta) = -\frac{1}{h^2} \begin{bmatrix} \alpha(e^{-i\theta_1} + e^{-i\theta_2} - 4) & h^2 \\ -h^2 & (e^{-i\theta_1} + e^{-i\theta_2} - 4) \end{bmatrix}$$

and

$$\tilde{\mathcal{A}}_h^-(\theta) = -\frac{1}{h^2} \begin{bmatrix} \alpha(e^{i\theta_1} + e^{i\theta_2}) & 0 \\ 0 & (e^{i\theta_1} + e^{i\theta_2}) \end{bmatrix}.$$

Therefore, the amplification factor is

$$\begin{aligned} \tilde{S}_h(\theta) &= -\tilde{\mathcal{A}}_h^+(\theta)^{-1} \tilde{\mathcal{A}}_h^-(\theta) \\ &= -\begin{bmatrix} -\frac{\alpha(e^{-i\theta_1} + e^{-i\theta_2} - 4)}{h^2} & -1 \\ 1 & -\frac{(e^{-i\theta_1} + e^{-i\theta_2} - 4)}{h^2} \end{bmatrix}^{-1} \begin{bmatrix} -\frac{\alpha(e^{i\theta_1} + e^{i\theta_2})}{h^2} & 0 \\ 0 & -\frac{(e^{i\theta_1} + e^{i\theta_2})}{h^2} \end{bmatrix}. \end{aligned}$$

DEFINITION 3.33 (Local smoothing factor)**.**
The local smoothing factor is defined by

$$\mu_{loc} = \mu_{loc}(S_h) := \sup\{|\rho(\tilde{S}_h(\theta))| : \theta \in T^{high}\}$$

where ρ denotes the spectral radius.

To quantify the smoothing factor, the eigenvalues of a 2×2 system can be determined by any symbolic package.

REMARK 3.34.

1. In [5], the following upper bound for the smoothing factor is given for the high frequencies for $h \in [0.01, 0.25]$ and $\alpha \in [10^{-6}, 1]$ for GS-LEX
$$\mu_{loc}(S_h) \leq 0.5.$$
2. The derivation of local smoothing factors for the ω-JAC can be derived in an analogous way.
3. The smoothing factor for GS-RB cannot be determined in that way as the smoothing operator cannot be splitted and therefore the assumptions of Lemma 3.32 are not fulfilled.

3.4.2.3 Two-grid Analysis

Another goal of the local Fourier analysis is to determine asymptotic two-grid convergence factors for the whole iteration TG_h^H. To this end, some preliminary work has to be done. As shown in the previous section, quadruples of $\varphi(\theta, \cdot)$ coincide on G_H. For any $\theta = (\theta_1, \theta_2) \in T^{low}$ consider the frequencies

$$\theta^{(0,0)} := (\theta_1, \theta_2), \qquad \theta^{(0,1)} := (\theta_1, \bar{\theta}_2)$$
$$\theta^{(1,0)} := (\bar{\theta}_1, \theta_2), \qquad \theta^{(1,1)} := (\bar{\theta}_1, \bar{\theta}_2)$$

where

$$\bar{\theta}_i := \begin{cases} \theta_i + \pi, \theta_i < 0 \\ \theta_i - \pi, \theta_i \geq 0. \end{cases}$$

This leads to

LEMMA 3.35.

1. For any frequency $\theta^{(0,0)} \in T^{low}$ it holds
$$\varphi(\theta^{(0,0)}, x) \equiv \varphi(\theta^{(1,1)}, x) \equiv \varphi(\theta^{(1,0)}, x) \equiv \varphi(\theta^{(0,1)}, x), \quad x \in G_H.$$
2. Each of these four Fourier components $\varphi(\theta^\beta, \cdot) = \varphi_h(\theta^\beta, \cdot)$ with $\beta \in \{(0,0), (1,1), (1,0), (0,1)\}$ coincide on G_H with the respective grid function $\varphi_H(2\theta^{(0,0)}, \cdot)$:
$$\varphi_h(\theta^\beta, x) \equiv \varphi_H(2\theta^{(0,0)}, x), \quad x \in G_H.$$

Proof. [53, Lemma 4.4.1]. □

DEFINITION 3.36 (Harmonics for standard coarsening)**.**

1. The corresponding four $\varphi(\theta^\beta, \cdot)$ are called harmonics.
2. For a given $\theta = \theta^{(0,0)} \in T^{low}$, define the four-dimensional space of harmonics by
$$E_h^\theta := span\left\{\varphi(\theta^\beta, \cdot) : \beta = (\beta_1, \beta_2) \in \{(0,0), (1,1), (1,0), (0,1)\}\right\}.$$

The spaces E_h^θ are invariant under the two-grid operator TG_h^H under general assumptions. An arbitrary $\psi \in E_h^\theta$ can be represented by

$$\psi = A^{(0,0)}\varphi(\theta^{(0,0)},\cdot) + A^{(1,1)}\varphi(\theta^{(1,1)},\cdot) + A^{(1,0)}\varphi(\theta^{(1,0)},\cdot) + A^{(0,1)}\varphi(\theta^{(0,1)},\cdot).$$

with uniquely determined coefficients A^β. For the case of the discretization of scalar differential equations $L_h u_h = f_h$ the following theorem holds.

THEOREM 3.37.

1. Let L_h, I_h^H, L_H and I_H^h be represented by stencils on G_h and G_H. Further let L_H^{-1} exist. Then the coarse grid correction operator CGC_h^H is represented on E_h^θ by the (4×4)-matrix
$$\widehat{CGC}_h^H(\theta) = \hat{I}_h - \hat{I}_H^h(\theta)(\hat{L}_H(2\theta))^{-1}\hat{I}_h^H(\theta)\hat{L}_h(\theta) \tag{3.12}$$
for each $\theta \in T^{low}$. Here, $\hat{I}_h, \hat{L}_h(\theta)$ are (4×4)-matrices, $\hat{I}_h^H(\theta)$ is a (4×1)-matrix, $\hat{L}_H(2\theta))^{-1}$ is a (1×1)-matrix and $\hat{I}_H^h(\theta)$ is a (1×4)-matrix.
2. If the spaces E_h^θ are invariant under the smoothing operation S_h, i.e.
$$S_h : E_h^\theta \to E_h^\theta \quad \forall\theta \in T^{low},$$
a representation of TG_h^H on E_h^θ by a (4×4)-matrix $\widehat{TG}_h^H(\theta)$ with respect to E_h^θ can be obtained by
$$\widehat{TG}_h^H(\theta) = \hat{S}_h(\theta)^{\nu_2}\widehat{CGC}_h^H(\theta)\hat{S}_h(\theta)^{\nu_1}$$
with $\widehat{CGC}_h^H(\theta)$ from (3.12) and the (4×4)-matrix $\hat{S}_h(\theta)$ which represents S_h.

Proof. [53, Theorem 4.4.1] □

By applying CGC_h^H to any $\psi \in E_h^\theta$, the coefficients A^β are transformed in the following way

$$\begin{pmatrix} A^{(0,0)} \\ A^{(1,1)} \\ A^{(1,0)} \\ A^{(0,1)} \end{pmatrix} \Leftarrow \widehat{CGC}_h^H(\theta) \begin{pmatrix} A^{(0,0)} \\ A^{(1,1)} \\ A^{(1,0)} \\ A^{(0,1)} \end{pmatrix}$$

and $TG_h^H\psi$ can be written as

$$TG_h^H\psi = B^{(0,0)}\varphi(\theta^{(0,0)},\cdot) + B^{(1,1)}\varphi(\theta^{(1,1)},\cdot) + B^{(1,0)}\varphi(\theta^{(1,0)},\cdot) + B^{(0,1)}\varphi(\theta^{(0,1)},\cdot)$$

with

$$\begin{pmatrix} B^{(0,0)} \\ B^{(1,1)} \\ B^{(1,0)} \\ B^{(0,1)} \end{pmatrix} = \widehat{TG}_h^H(\theta) \begin{pmatrix} A^{(0,0)} \\ A^{(1,1)} \\ A^{(1,0)} \\ A^{(0,1)} \end{pmatrix}.$$

Now the local Fourier analysis is applied to estimate rates of convergence for the multigrid method solving the optimality system $\mathcal{A}_h w_h = \phi_w$. The two-grid operator is given by

$$TG_h^H = S_h^{\nu_2}[I_h - I_H^h(\mathcal{A}_H)^{-1}I_h^H\mathcal{A}_h]S_h^{\nu_1}$$

and the coarse grid corrector by

$$CG_h^H = [I_h - I_H^h(\mathcal{A}_H)^{-1} I_h^H \mathcal{A}_h].$$

For studying the action of TG_h^H on an arbitrary couple $(\epsilon_y, \epsilon_p) \in E_h^\theta \times E_h^\theta$, define

$$\epsilon_y = \sum_\beta Y^\beta \varphi(\theta^\beta, \cdot) \quad \text{and} \quad \epsilon_p = \sum_\beta P^\beta \varphi(\theta^\beta, \cdot)$$

where (ϵ_y, ϵ_p) represents the error functions for y_h and p_h.
For each θ, the spaces $E_h^\theta \times E_h^\theta$ are invariant, as shown above. The next theorem is a result by Borzì, Kunisch and Kwak [5, Theorem 5.1], an extension of the Theorem 3.37 applied to the optimality system.

THEOREM 3.38.

1. Under the assumptions that all multigrid components in TG_h^H are linear and that $(\mathcal{A}_H)^{-1}$ exists, the coarse grid operator CG_h^H is represented on $E_h^\theta \times E_h^\theta$ by the (8×8)-matrix $\widehat{CG}_h^H(\theta)$,
$$\widehat{CG}_h^H(\theta) = [\hat{I}_h - \hat{I}_H^h(\theta)(\hat{\mathcal{A}}_H(2\theta))^{-1}\hat{I}_h^H(\theta)\hat{\mathcal{A}}_h(\theta)],$$
for each $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)^2$. Here, $\hat{I}_h$ and $\hat{A}_h(\theta)$ are (8×8)-matrices, $\hat{I}_h^H(\theta)$ is a (2×8)-matrix, $\hat{I}_H^h(\theta)$ is a (8×2)-matrix and $\hat{\mathcal{A}}_H(2\theta)$ is a (2×2)-matrix.
2. If the spaces $E_h^\theta \times E_h^\theta$ are invariant under the smoothing operator S_h, i.e. $\hat{S}_h(\theta) : E_h^\theta \times E_h^\theta \to E_h^\theta \times E_h^\theta$ for all $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)^2$, then $\hat{S}_h(\theta)$ is a (8×8)-matrix and a representation of TG_h^H on $E_h^\theta \times E_h^\theta$ by a (8×8)-matrix is given by
$$\widehat{TG}_h^H(\theta) = \hat{S}_h(\theta)^{\nu_2}\widehat{CG}_h^H(\theta)\hat{S}_h(\theta)^{\nu_1}.$$

In explicit form, the symbols of the operators are

- for the coarse grid operator $\mathcal{A}_H$:
$$\hat{A}_H(2\theta) = \begin{bmatrix} \alpha\frac{4-2\cos(2\theta_1)+\cos(2\theta_2)}{H^2} & -1 \\ 1 & \frac{4-2\cos(2\theta_1)+\cos(2\theta_2)}{H^2} \end{bmatrix}$$
- for the fine grid operator $\mathcal{A}_h$:
$$\hat{\mathcal{A}}_h(\theta) = \begin{bmatrix} \alpha\xi(\theta^{(0,0)}) & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & \alpha\xi(\theta^{(1,1)}) & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & \alpha\xi(\theta^{(1,0)}) & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & \alpha\xi(\theta^{(0,1)}) & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & \xi(\theta^{(0,0)}) & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \xi(\theta^{(1,1)}) & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \xi(\theta^{(1,0)}) & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \xi(\theta^{(0,1)}) \end{bmatrix}$$
where
$$\xi(\theta^\beta) = \frac{4 - 2(\cos(\theta_1^{\beta_1}) + \cos(\theta_2^{\beta_2}))}{h^2}$$

- for the restriction operator I_h^H:

$$\hat{I}_h^H(\theta) = \begin{bmatrix} I(\theta^{(0,0)}) & I(\theta^{(1,1)}) & I(\theta^{(1,0)}) & I(\theta^{(0,1)}) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I(\theta^{(0,0)}) & I(\theta^{(1,1)}) & I(\theta^{(1,0)}) & I(\theta^{(0,1)}) \end{bmatrix}$$

  where

$$I(\theta^\beta) = I_h^H(\theta^\beta) = \frac{1}{4}(1+\cos(\theta_1^{\beta_1}))(1+\cos(\theta_2^{\beta_2})).$$

- prolongation operator I_H^h:

$$\hat{I}_H^h(\theta) = \hat{I}_h^H(\theta)^T.$$

DEFINITION 3.39 (Asymptotic convergence factor)**.**
The asymptotic convergence factor is defined by

$$\rho_{loc} = \rho_{loc}(\widehat{TG}_h^H) = \sup\left\{\rho(\widehat{TG}_h^H(\theta)) : \theta \in T^{low}\right\}$$

where $\rho(\widehat{TG}_h^H(\theta))$ is the spectral radius of $\widehat{TG}_h^H(\theta)$.

The 8×8 matrix $\widehat{TG}_h^H(\theta)$ corresponds to the error function and its components (Y^β, P^β) and expresses the action of one TG_h^H step by

$$(Y^\beta, P^\beta)^{(1)} = \widehat{TG}_h^H(\theta)(Y^\beta, P^\beta)^{(0)}.$$

In Theorem 3.38, a more general requirement on the smoothing operator instead of the splitting property was proven. The invariance property

$$S_h : E_h^\theta \times E_h^\theta \to E_h^\theta \times E_h^\theta \quad \text{for all } \theta \in T^{low}$$

has to hold. This assumption is fulfilled for the GS-RB. For smoother satisfying the invariance property, high and low frequency may be intermixed. Therefore an *ideal coarse grid operator* Q_h^H is defined, which annihilates the low frequency components and leaves the high frequency components unchanged. Define on E_h^θ

$$Q_h^H\varphi(\theta,\cdot) := \begin{cases} 0 & \text{if } \theta = \theta^{(0,0)} \in T^{low} \\ \varphi(\theta,\cdot) & \text{if } \theta \in \{\theta^{(1,0)}, \theta^{(0,1)}, \theta^{(1,1)}\} \end{cases}$$

and on the space $E_h^\theta \times E_h^\theta$

$$\hat{Q}_h^H(\theta) := \begin{bmatrix} Q_h^H & 0 \\ 0 & Q_h^H \end{bmatrix} \quad \text{for } \theta \in T^{low}.$$

DEFINITION 3.40.
Under the assumption that S_h has the invariance property, the smoothing factor $\bar{\mu}_{loc}(S_h, \nu)$ of S_h is defined by

$$\bar{\mu}_{loc}(S_h, \nu) := \sup\left\{\sqrt[\nu]{\rho(\hat{Q}_h^H \hat{S}_h(\theta)^\nu)}, \theta \in T^{low}\right\}.$$

	LFA		Exp.
$(\nu_1+\nu_2)$	$\bar{\mu}_{loc}(S_h, \nu_1+\nu_2)$	$\rho_{loc}(\widehat{TG}_h^H)$	V(ν_1,ν_2)
(1+1)	0.25	0.25	0.30
(2+1)	0.125	0.12	0.12
(2+2)	0.06	0.08	0.08
(3+2)	0.03	0.06	0.06
(3+3)	0.01	0.05	0.05

Table 3.1: Convergence Factors

Here, $\bar{\mu}$ depends on ν. This is another definition of the smoothing operator, assuming that an ideal CGC operator is defined.
Borzì et al. reported in [5] the values $\bar{\mu}(S_k, \nu)$ and ρ_{loc} for the two-grid multigrid algorithm with the GS-LEX smoothing iteration for $h \in [0.01, 0.25]$ and $\alpha \in [10^{-6}, 1]$, compared with experimental values, where the convergence factor was measured as the ratio of the discrete L^2-norm of residuals resulting of two successive multigrid cycles.

These values are typical for the standard scalar Poisson problem.

REMARK 3.41.
A generalization of the local Fourier analysis for scalar equations and systems of PDEs with nonconstant coefficients or nonlinear systems can be applied by linearizing and freezing of coefficients.

3.4.3 CSMG Convergence Theory

Whereas the local Fourier analysis was used to derive sharp convergence estimates, in this section the multigrid convergence to weak solutions of the optimality system is proven. Starting first with the multigrid convergence framework by Hackbusch [29] for the scalar uniformly elliptic partial differential equations with the finite differences discretization on a grid Ω_k, as defined in Section 3.3.1 by

$$A_k u_k = f_k.$$

In the second part, a slight variation of these proofs is used to show multigrid convergence of the optimality system. A proof for the collective smoothing multigrid method with the Jacobi smoother can be found in [51].

3.4.3.1 Smoothing and Approximation Property

In this section, sufficient conditions for the convergence of the linear multigrid method are discussed: the smoothing property and the approximation property.

DEFINITION 3.42 (Smoothing property)**.**
An iteration S_k^ν is said to possess the smoothing property if there exist functions $\eta(\nu)$ and $\bar{\nu}(h)$ and a scalar $\alpha > 0$ such that

1. $\left\|A_k S_k^\nu\right\| \leq \eta(\nu) h_k^{-\alpha}$ for all $1 \leq \nu < \bar{\nu}(h_k), k \geq 1$,
2. $\lim_{\nu\to\infty} \eta(\nu) = 0$,
3. $\lim_{h\to\infty} \bar{\nu}(h) = \infty$ or $\bar{\nu}(h) = \infty$.

The functions η and $\bar{\nu}$ are required to be independent of k or h_k.

Hackbusch [29, Chapter 6.2] proved that the smoothing property holds for the Richardson iteration, the damped Jacobi iteration, the two-cyclic Gauss-Seidel iteration (and here in particular the Gauss-Seidel iteration with chequer-board ordering for the five-point stencil) and the Kaczmarz-iteration.

DEFINITION 3.43 (Approximation property)**.**
The approximation property holds if there is some constant C_A such that

$$\left\|A_k^{-1} - I_{k-1}^k A_{k-1}^{-1} I_k^{k-1}\right\| \leq C_A h_k^\alpha \quad \text{for all } k \geq 1.$$

Whilst the smoothing property is an algebraic one, the smoothing property depends on the properties of the boundary value problem, see [29, Chapter 6.3]. With these two assumptions, the general convergence of the two-grid multigrid can be shown.

THEOREM 3.44 (General two-grid convergence)**.**
Let the smoothing property and the approximation property hold and let $0 < \zeta < 1$ be a fixed number.

1. In the case $\bar{\nu}(h) = \infty$ there is a lower bound $\underline{\nu}$ such that the the two-grid contraction number satisfies
$$\left\|M_k^{TGM(\nu,0)}\right\| \leq C_A \eta(\nu) \leq \zeta \tag{3.13}$$
for $\nu \geq \underline{\nu}$ and $k \geq 1$.
2. If $\bar{\nu}(h) \to \infty$ there are bounds $\bar{h} > 0$ and $\underline{\nu}$ such that the inequality (3.13) holds for all $\nu \in [\underline{\nu}, \bar{\nu}(h))$ and all k with $h_k \leq \bar{h}$. For such k, the interval $[\underline{\nu}, \bar{\nu}(h_k))$ is not empty.

Proof. [29, Theorem 6.1.7]. □

For proving the convergence of the multigrid iteration, two further propositions have to be made:

$$\begin{aligned} &1. \quad \|S_k^\nu\| \leq C_S \quad \text{for all } k \geq 1, 0 < \nu < \bar{\nu} := \min_{k\geq 1} \bar{\nu}(h_k) \\ &2. \quad \underline{C}_P^{-1} \|u_{k-1}\| \leq \left\|I_k^{k-1} u_{k-1}\right\| \leq \bar{C}_P \|u_{k-1}\| \quad \text{for all } u_{k-1} \in U_{k-1}, k \geq 1 \end{aligned} \tag{3.14}$$

THEOREM 3.45 (Multigrid convergence)**.**
Suppose the cycle-index $\gamma \geq 2$, let the smoothing property and the approximation property hold just as well as the two assumptions (3.14). Let $\zeta' \in (0, 1)$ be a fixed number.

1. In the case $\bar{\nu}(h) = \infty$ there is a lower bound $\underline{\nu}$ such that the multigrid contraction number satisfies
$$\left\|M_k^{MGM(\nu,0)}\right\| \leq \zeta' < 1, \quad \left\|M_k^{MGM(\nu,0)}\right\| \leq \frac{\gamma}{\gamma - 1} C_A \eta(\nu),$$
whenever $\nu \geq \underline{\nu}$, independently of $k \geq 1$.

2. If $\bar{\nu} \to \infty$ there exist $\bar{h} > 0$ and $\underline{\nu}$ such that (1) holds for all $\nu \in [\underline{\nu}, \bar{\nu}(h_k))$ and all $k \geq 1$, provided that $h_k \leq \bar{h}$. For such h_k the interval $[\underline{\nu}, \bar{\nu}(h_k))$ is not empty.

Proof. [29, Theorem 7.1.2]. □

3.4.3.2 Multigrid Convergence

Now the multigrid convergence for the optimality system is presented. Following [7] and [5], the proof is split into three parts. In a first step, convergence is shown for

$$\begin{aligned} -\Delta y &= f \quad \text{in } \Omega \\ y &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

In the second step, the decoupled symmetric system

$$\begin{aligned} -\alpha\Delta y &= \alpha f \quad \text{in } \Omega \\ y &= 0 \quad \text{on } \partial\Omega \\ -\Delta p &= y_d \quad \text{in } \Omega \\ p &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

is considered. In the final step, the multigrid convergence for the nonsymmetric optimality system is proven. This multigrid convergence theory goes back to [12]. For this purpose, the Algorithm 3.1 is reformulated in the form of a classical iteration scheme as

$$M_k = I_k - B_k A_k,$$

where I_k denotes the identity on V_k. Let $R_k : V_k \to V_k$ be an iteration operator such that $S_k = I_k - R_k A_k$ for $k > 1$. Let

$$A_k u_k = f_k \quad \text{in } V_k$$

be the matrix form of the discretization of

$$\begin{aligned} -\Delta y &= f \quad \text{in } \Omega \\ y &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

Introduce for variables u on V_k the inner product $(\cdot,\cdot)_k$ with associated norm $\|u\|_k = (u,u)_k^{1/2}$ and define $I_k^{k-1} : V_k \to V_{k-1}$ as the L_k^2-projection described by

$$(I_k^{k-1} u, v)_{k-1} = (u, I_{k-1}^k v)_k \quad \forall u \in V_k, v \in V_{k-1}$$

and $P_{k-1} : V_k \to V_{k-1}$ as the A_k-projection by

$$(A_{k-1} P_{k-1} u, v)_{k-1} = (A_k u, I_{k-1}^k v)_k \quad \forall u \in V_k, v \in V_{k-1}.$$

With these definitions, the V-cycle multigrid algorithm is given in the recursive form as in [7]. The equivalence of the recursive form to the Algorithm 3.1 is shown in [7]. According to [12], the following multigrid convergence theorem is proven.

Algorithm 3.3 Multigrid scheme: recursive form

Set $B_1 = A_1^{-1}$. For $k \geq 2$ define $B_k : V_k \to V_k$ in terms of B_{k-1} as follows. Let $f_k \in V_k$.

1: Define $u^{(l)}$ for $l = 1, \ldots, \nu_1$ by

$$u^{(l)} = u^{(l-1)} + R_k(f_k - A_k u^{(l-1)}).$$

2: Set $u^{(\nu_1+1)} = u^{(\nu_1)} + I_{k-1}^k q$, where

$$q = B_{k-1} I_k^{k-1}(f_k - A_k u^{(\nu_1)}).$$

3: Set $B_k f_k = u^{(\mu_1+\mu_2+1)}$, where $u^{(l)}$ for $l = \nu_1 + 2, \ldots, \nu_1 + \nu_2 + 1$ is given by Step 2 (with R_k^T instead of R_k for a symmetric multigrid scheme).

THEOREM 3.46 (Multigrid convergence for scalar equations)**.**
Let R_k satisfy the assumptions

1. Smoothing operator R_k:
   There exist constants $C_R > 0$ and $c > 0$ independent of u and k such that

$$C_R \frac{\|u\|_k^2}{\lambda_k} \leq (Ru, u)_k \leq c(A_k^{-1}u, u)_k \quad \forall u \in V_k,$$

   where λ_k denotes the largest eigenvalue of A_k.

2. Regularity and approximation assumption:
   There exist $0 < \alpha \leq 1$ and a constant C_α independent of k such that

$$\left(A_k \left(I_k - I_{k-1}^k P_{k-1}\right) u, u\right)_k \leq C_\alpha \left(\frac{\|A_k u\|_k^2}{\lambda_k}\right)^\alpha (A_k u, u)_k^{1-\alpha} \quad \forall u \in V_k$$

   for $k > 1$.

Then there exists a positive constant $\delta_k < 1$ such that

$$(A_k M_k u, u)_k \leq \delta_k (A_k u, u)_k \quad \forall u \in V_k,$$

where $M_k = I_k - B_k A_k$ and $\delta_k = \delta = \frac{C_1}{C_1 + 2\nu C_R}$ with $0 < \delta < 1$ and $\nu = \nu_1 + \nu_2 \geq 1$.

Proof. [7, Theorem 5.7]. □

In a second step, consider the decoupled symmetric system

$$\begin{aligned} -\alpha \Delta y &= \alpha f && \text{in } \Omega \\ y &= 0 && \text{on } \partial\Omega \\ -\Delta p &= y_d && \text{in } \Omega \\ p &= 0 && \text{on } \partial\Omega. \end{aligned}$$

As these are exactly two copies of the Poisson equation, the multigrid convergence theory preserves the properties of the first case. Define

$$\hat{A}_k = \begin{bmatrix} \alpha A_k & 0 \\ 0 & A_k \end{bmatrix}$$

and analogously $\hat{B}_k$ and $\hat{M}_k$. Let $w_k = (y_k, p_k) \in V_k \times V_k =: V_k^2$.

THEOREM 3.47 (Multigrid convergence for decoupled system)**.**
Under the assumptions of Theorem 3.46 there exists a positive constant $\delta < 1$ such that

$$(\hat{A}_k \hat{M}_k w, w)_k \leq \delta (\hat{A}_k w, w)_k,$$

with the same δ as in Theorem 3.46.

The last step is to define

$$\mathcal{A}_k = \hat{A}_k + d_k,$$

where

$$d_k = \begin{bmatrix} 0 & -I_k \\ I_k & 0 \end{bmatrix}.$$

The multigrid algorithm corresponding to this nonsymmetric problem has exactly the same structure as in Algorithm 3.3 with B_k, A_k, M_k replaced by $\mathcal{B}_k, \mathcal{A}_k, \mathcal{M}_k$ and thus

$$\mathcal{M}_k = I_k - \mathcal{B}_k \mathcal{A}_k = \left[I_k - I_{k-1}^k \mathcal{P}_{k-1} + I_{k-1}^k \left(I_{k-1} - \mathcal{B}_{k-1} \mathcal{A}_{k-1} \right) P_{k-1} \right] S_k,$$

where I_k denotes the identity operator on V_k.

THEOREM 3.48 (Multigrid convergence for coupled system)**.**
There exist positive constants h_0 and $\tilde{\delta} < 1$ such that for all $h_1 < h_0$

$$(\mathcal{A}_k \mathcal{M}_k w, w)_k \leq \tilde{\delta} (\mathcal{A}_k w, w)_k \quad \forall w \in V_k^2,$$

where $\tilde{\delta} = \delta + C h_1$ and δ as in Theorem 3.46.

Proof. [7, Theorem 5.19]. □

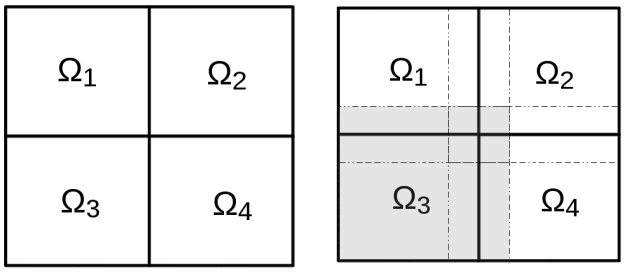
The constant δ depends on the features of the optimality system, such as for example non-symmetry. For a sufficiently small h_1, there holds $\tilde{\delta} \approx \delta$ and the convergence factor of the multigrid scheme applied to the optimality system is close to the convergence factor of the multigrid scheme applied to the scalar Poisson problem.

4 Domain Decomposition

In this chapter a nonoverlapping domain decomposition method for the solution of the optimality system is introduced. In Section 4.1 a brief overview of the idea and the theory of domain decompositions, followed by a two subdomain formulation with the definition of the Steklov-Poincaré operator on the interface as well as a discrete approximation with the Schur complement method is given in Section 4.2. The exact decomposition of the Schur complement for the two-dimensional case is derived in Section 4.3 and its inverse is determined in Section 4.4. Some straightforward extensions for the multi-domain case and for the case of three dimensions are proposed in Section 4.5 and in Section 4.6, respectively. At last, the use as a preconditioner of the Schur complement for more general differential operators on irregular domains is suggested.

4.1 Domain Decomposition Methods

The key idea of domain decompositions for the numerical solution of partial differential equations is to split the entire spatial domain Ω into several smaller subdomains Ω_i, $i = 1, \ldots, N$, $N \geq 2$. Upon every subdomain Ω_i, the original problem is reformulated. This yields to a family of subproblems of reduced size that are interconnected through the values of the unknown solution at the subdomain interfaces and possibly in a set of cross-points. The interface coupling is removed by introducing an iterative process among subdomains, which leads to independent subproblems upon subdomains at each step. These techniques can often be applied directly to the partial differential equations, but they are of most interest when applied to discretizations of the differential equations, as realized in this thesis.
There are two main approaches, illustrated in Figure 4.1: on the one hand, *overlapping domain decomposition methods* are proposed, where the subdomains overlap by more than the interface, and on the other hand, *nonoverlapping domain decomposition methods* exist, where the subdomains only intersect on their interfaces.



(a) nonoverlapping (b) overlapping

Figure 4.1: Domain Decomposition

For the nonoverlapping domain decomposition, the continuity of the solution across subdomain interfaces is enforced by representing the value of the solution on all neighboring subdomains by the same unknown. These methods are also called *iterative substructuring*. After determining the interface unknowns, the solution at the interior of the subdomains can be computed by solving independent problems with the computed interface values as boundary conditions. The pioneering work stems from Schwarz in the end of the 19th century [50]. A detailed survey can be found in [19].
For general domain decomposition methods, the rate of convergence depends on the amount of overlapping: greater overlapping implies faster convergence, but more effort per iteration since the work is duplicated on the overlapping domain.
Domain decomposition methods are applied in many situations: as communication is limited to the interfaces of the subdomains, the problem can be decoupled into independent subproblems and therefore allows the processing in parallel on multiprocessor systems. Even in sequential computer environments, it can be adapted for irregular domains, where a natural partition into regular subdomains is given (and a fast solver exists), or for dividing a problem with discontinuous coefficients into subregions with constant coefficients. For computers with limited memory, domain decomposition methods are often the only possibility to deal with very large problems.

4.2 Two-Subdomain Formulation

To begin with, a formulation for the case of a decomposition into two nonoverlapping subdomains is given. For the continuous formulation of the optimality system, the *Steklov-Poincaré operator* is defined in Section 4.2.1 to derive transmission conditions to ensure continuity of the solution and the fluxes across the interface. A discrete approximation of this operator is determined in Section 4.2.2 with the *Schur complement method*.

4.2.1 Steklov-Poincaré Operator

The partition of Ω into two nonoverlapping subdomains Ω_1, Ω_2 with the interface $B = \partial\Omega_1 \cap \Omega (= \partial\Omega_2 \cap \Omega)$ is considered, as illustrated in Figure 4.2.

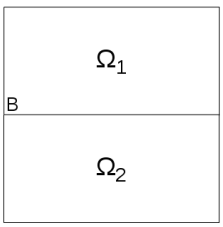


Figure 4.2: Two subdomains

The problem on the entire domain is given by the coupled system of elliptic PDEs written in

operator-matrix form

$$\begin{aligned} \begin{bmatrix} \alpha L & -I \\ I & L \end{bmatrix} \begin{pmatrix} y \\ p \end{pmatrix} &= \begin{pmatrix} \alpha f \\ y_d \end{pmatrix} \quad \text{in } \Omega \\ \begin{pmatrix} y \\ p \end{pmatrix} &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \text{on } \partial\Omega \end{aligned} \tag{4.1}$$

as introduced in Chapter 3. With the notation

$$\mathcal{A} := \begin{bmatrix} \alpha L & -I \\ I & L \end{bmatrix}, \quad w := \begin{pmatrix} y \\ p \end{pmatrix} \quad \text{and} \quad \phi := \begin{pmatrix} \alpha f \\ y_d \end{pmatrix}, \tag{4.2}$$

the problem formulation is given by

$$\begin{aligned} \mathcal{A}w &= \phi \quad \text{in } \Omega \\ w &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

Let $w = (w_1, w_2, w_B)$ and $\phi = (\phi_1, \phi_2, \phi_B)$ denote the restriction of w and ϕ to Ω_1,Ω_2 and B, respectively. Then the overall problem can be reformulated as the two local problems

$$\begin{aligned} \mathcal{A}w_1 &= \phi_1 \quad \text{in } \Omega_1 \\ w_1 &= 0 \quad \text{on } \partial\Omega_1 \setminus B \\ w_1 &= w_B \quad \text{on } B \end{aligned} \tag{4.3}$$

and

$$\begin{aligned} \mathcal{A}w_2 &= \phi_2 \quad \text{in } \Omega_2 \\ w_2 &= 0 \quad \text{on } \partial\Omega_2 \setminus B \\ w_2 &= w_B \quad \text{on } B \end{aligned} \tag{4.4}$$

with the transmission boundary conditions on the continuity of the flux across B

$$\begin{aligned} \frac{\partial y_1}{\partial n_1} &= -\frac{\partial y_2}{\partial n_2} \\ \frac{\partial p_1}{\partial n_1} &= -\frac{\partial p_2}{\partial n_2}, \end{aligned}$$

where n_i denotes the outward pointing normal vector to B from Ω_i.
The transmission conditions are derived by the equivalence of the weak formulations of the problem formulation on the two subspaces and the entire problem.

REMARK 4.1.
The weak formulation of (4.2) is given by

$$\begin{aligned} \int_\Omega \alpha L y v - p w \, \mathrm{dV} &= \int_\Omega \alpha f v \, \mathrm{dV} \quad \forall v, w \in H_0^1(\Omega) \\ \int_\Omega y v + L p w \, \mathrm{dV} &= \int_\Omega y_d w \, \mathrm{dV}. \end{aligned} \tag{4.5}$$

Integrating by parts, this is equivalent to

$$\begin{aligned} -\alpha \int_\Omega \nabla y \nabla v \, \mathrm{dV} - \int_\Omega p w \, \mathrm{dV} &= \int_\Omega \alpha f v \, \mathrm{dV} \quad \forall v, w \in H_0^1(\Omega) \\ \int_\Omega y v \, \mathrm{dV} - \int_\Omega \nabla p \nabla w \, \mathrm{dV} &= \int_\Omega y_d w \, \mathrm{dV}. \end{aligned} \tag{4.6}$$

Conversely, splitting the integrals in (4.6) proves that this is equivalent to

$$\begin{aligned} -\alpha \int_{\Omega_1} \nabla y_1 \nabla v \, \mathrm{dV} - \alpha \int_{\Omega_2} \nabla y_2 \nabla v \, \mathrm{dV} - \int_{\Omega_1} p_1 w \, \mathrm{dV} - \int_{\Omega_2} p_2 w \, \mathrm{dV} \\ = \int_{\Omega_1} \alpha f_1 v \, \mathrm{dV} + \int_{\Omega_2} \alpha f_2 \, \mathrm{dV} \quad \forall v, w \in H_0^1(\Omega) \\ \int_{\Omega_1} y_1 v \, \mathrm{dV} + \int_{\Omega_2} y_2 v \, \mathrm{dV} - \int_{\Omega_1} \nabla p_1 \nabla w \, \mathrm{dV} - \int_{\Omega_2} \nabla p_2 \nabla w \, \mathrm{dV} \\ = \int_{\Omega_1} y_{d1} w \, \mathrm{dV} + \int_{\Omega_2} y_{d2} w \, \mathrm{dV}. \end{aligned} \tag{4.7}$$

The weak formulation of (4.3) is given by

$$\begin{aligned} \int_{\Omega_1} \alpha L y_1 v - p_1 w \, \mathrm{dV} = \int_{\Omega_1} \alpha f_1 v \, \mathrm{dV} \quad \forall v, w \in H_0^1(\Omega) \\ \int_{\Omega_1} y_1 v + L p_1 w \, \mathrm{dV} = \int_{\Omega} y_{d1} w \, \mathrm{dV} \end{aligned} \tag{4.8}$$

and again integrating by parts, this is equivalent to

$$\begin{aligned} \alpha \int_{\Omega_1} \nabla y_1 \nabla v \, \mathrm{dV} - \int_{\Omega_1} p_1 w \, \mathrm{dV} + \int_B v \frac{\partial y_1}{\partial n_1} \, \mathrm{dA} = \int_{\Omega_1} \alpha f_1 v \, \mathrm{dV} \quad \forall v, w \in H_0^1(\Omega) \\ \int_{\Omega_1} y_1 v \, \mathrm{dV} - \int_{\Omega_1} \nabla p_1 \nabla w \, \mathrm{dV} + \int_B w \frac{\partial p_1}{\partial n_1} \, \mathrm{dA} = \int_{\Omega_1} y_{d1} w \, \mathrm{dV}. \end{aligned} \tag{4.9}$$

Analogously for (4.4), it holds

$$\begin{aligned} -\alpha \int_{\Omega_2} \nabla y_2 \nabla v \, \mathrm{dV} - \int_{\Omega_2} p_2 w \, \mathrm{dV} + \int_B v \frac{\partial y_2}{\partial n_2} \, \mathrm{dA} = \int_{\Omega_2} \alpha f_2 \, \mathrm{dV} \quad \forall v, w \in H_0^1(\Omega) \\ \int_{\Omega_2} y_2 v \, \mathrm{dV} - \int_{\Omega_2} \nabla p_2 \nabla w \, \mathrm{dV} + \int_B w \frac{\partial p_2}{\partial n_2} \, \mathrm{dA} = \int_{\Omega_2} y_{d2} w \, \mathrm{dV}. \end{aligned} \tag{4.10}$$

Comparing (4.7) with (4.9) and (4.10), the conditions

$$\int_B v \left(\frac{\partial y_1}{\partial n_1} + \frac{\partial y_2}{\partial n_2} \right) \mathrm{dA} = 0 \quad \text{and} \quad \int_B w \left(\frac{\partial p_1}{\partial n_1} + \frac{\partial p_2}{\partial n_2} \right) \mathrm{dA} = 0 \quad \forall v, w \in H_0^1(\Omega)$$

have to be fulfilled. Therefore,

$$\frac{\partial y_1}{\partial n_1} = -\frac{\partial y_2}{\partial n_2} \text{ and } \frac{\partial p_1}{\partial n_1} = -\frac{\partial p_2}{\partial n_2}.$$

For the treatment of more general boundary conditions see [4].
The main task is to determine the unknown values w_B on the interface. If the solution w_B on B is known, then the local solutions on Ω_1 and Ω_2 can be obtained by solving the two subproblems with Dirichlet boundary conditions, (4.3) and (4.4). This can be processed in parallel. For this purpose, an equation to identify w_B can be obtained by using the transmission boundary

conditions. According to [19], let g denote arbitrary Dirichlet boundary data on B and define $E_1 g$ and $E_2 g$ as solutions of the local problems

$$\begin{aligned} \mathcal{A}(E_1 g) &= \phi_1 \quad \text{in } \Omega_1 \\ E_1 g &= 0 \quad \text{on } \partial\Omega_1 \setminus B \\ E_1 g &= g \quad \text{on } B \end{aligned}$$

and

$$\begin{aligned} \mathcal{A}(E_2 g) &= \phi_2 \quad \text{in } \Omega_2 \\ E_2 g &= 0 \quad \text{on } \partial\Omega_2 \setminus B \\ E_2 g &= g \quad \text{on } B. \end{aligned}$$

The boundary values of $E_1 g$ and $E_2 g$ match on B (by construction), but the flux of the two local solutions will only match on B if

$$\begin{aligned} \frac{\partial y_1}{\partial n_1} &= -\frac{\partial y_2}{\partial n_2} \quad \text{on } B \\ \frac{\partial p_1}{\partial n_1} &= -\frac{\partial p_2}{\partial n_2} \quad \text{on } B, \end{aligned}$$

i.e. if $g = w_B$.
Next, an operator T is defined which maps the boundary data g on B to the jump in the flux across B.

DEFINITION 4.2 (Steklov-Poincaŕe operator)**.**
The Steklov-Poincaré operator T is given by

$$T : g \to \begin{pmatrix} \frac{\partial y_1}{\partial n_1} + \frac{\partial y_2}{\partial n_2} \\ \frac{\partial p_1}{\partial n_1} + \frac{\partial p_2}{\partial n_2} \end{pmatrix}.$$

The Steklov-Poincaré operator T is a pseudo-differential operator which is an affine linear mapping T and positive definite but not symmetric as $\mathcal{A}$ is not symmetric.
The equation $T w_B = 0$ is satisfied for the boundary value w_B of the true solution w. For further reading, see [1] and [45].

4.2.2 Schur Complement Method

A discrete approximation of the Steklov-Poincaré operator T can be obtained by the *Schur complement method*. In this method, the discretized system of partial differential equations is factorized and a system for the unknowns on the interface between these subregions is derived. This interface system is also called *capacitance system*. Consider the coupled system of partial differential equations

$$\mathcal{A} w = \phi$$

on the rectangular domain Ω, partitioned into two subdomains Ω_1 and Ω_2 with the common interface B. The discretized system

$$\mathcal{A}_h w_h = \phi_h$$

is modified analogously to the continuous case: let Ω be partitioned in $\Omega = \Omega_1 \cup \Omega_2 \cup B$ and $I = I_1 \cup I_2 \cup I_B$ be the partition of the indices in the linear system. I_1, I_2 are the indices of the nodes in Ω_1 and Ω_2, respectively. I_B consists of the nodes on the interface B. The notation $w_h = [w_1, w_2, w_B]^T$ and $\phi_h = [\phi_1, \phi_2, \phi_B]^T$ is understood accordingly. After a rearrangement of the unknowns, the linear system is given in the formulation

$$\begin{bmatrix} \mathcal{A}_1 & \mathcal{A}_{12} & \mathcal{A}_{1B} \\ \mathcal{A}_{21} & \mathcal{A}_2 & \mathcal{A}_{2B} \\ \mathcal{A}_{1B}^T & \mathcal{A}_{2B}^T & \mathcal{A}_B \end{bmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_B \end{pmatrix} = \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_B \end{pmatrix}.$$

When using a low-order finite difference discretization for the uniformly elliptic operator L where only one neighbored grid point in each direction is taken into account, the only relation between the nodes in Ω_1 and Ω_2 is through the interface. Therefore, the blocks $\mathcal{A}_{12}$ and $\mathcal{A}_{21}$ are zero matrices and thus

$$\mathcal{A}_h = \begin{bmatrix} \mathcal{A}_1 & 0 & \mathcal{A}_{1B} \\ 0 & \mathcal{A}_2 & \mathcal{A}_{2B} \\ \mathcal{A}_{1B}^T & \mathcal{A}_{2B}^T & \mathcal{A}_B \end{bmatrix}.$$

The matrix $\mathcal{A}_h$ is factorized in the following manner

$$\mathcal{A}_h = \begin{bmatrix} \mathcal{A}_1 & 0 & 0 \\ 0 & \mathcal{A}_2 & 0 \\ \mathcal{A}_{1B}^T & \mathcal{A}_{2B}^T & S \end{bmatrix} \begin{bmatrix} I & 0 & \mathcal{A}_1^{-1}\mathcal{A}_{1B} \\ 0 & I & \mathcal{A}_2^{-1}\mathcal{A}_{2B} \\ 0 & 0 & I \end{bmatrix}$$

where S is the Schur complement of $\mathcal{A}_B$ in $\mathcal{A}_h$, i.e.,

$$S = \mathcal{A}_B - \mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\mathcal{A}_{1B} - \mathcal{A}_{2B}^T\mathcal{A}_2^{-1}\mathcal{A}_{2B}.$$

The problem to find the unknowns on the interface B is formulated by

$$S w_B = \tilde{\phi}_B,$$

where $\tilde{\phi}_B = \phi_B - \mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\phi_1 - \mathcal{A}_{2B}^T\mathcal{A}_2^{-1}\phi_2$. The equation $S w_B - \tilde{\phi}_B = 0$ is the discrete approximation of the Steklov-Poincaré equation $T w_B = 0$, which enforces the transmission boundary conditions. The Schur complement algorithm to determine the interface grid points is formulated in Algorithm 4.1.

Algorithm 4.1 Schur complement algorithm for two subdomains

Let Ω be decomposed into two subdomains Ω_1 and Ω_2.

1: Compute $-\mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\phi_1$ and $-\mathcal{A}_{2B}^T\mathcal{A}_2^{-1}\phi_2$.
2: Set $\tilde{\phi}_B = \phi_B - \mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\phi_1 - \mathcal{A}_{2B}^T\mathcal{A}_2^{-1}\phi_2$.
3: Solve $S w_B = \tilde{\phi}_B$.
4: Compute $w_1 = \mathcal{A}_1^{-1}(\phi_1 - \mathcal{A}_{1B} w_B)$ and $w_2 = \mathcal{A}_2^{-1}(\phi_2 - \mathcal{A}_{2B} w_B)$.
5: Set $w = [w_1, w_B, w_2]^T$.

The right-hand side setup in Step 1 can be evaluated by solving two subdomain problems, one on Ω_1 and one on Ω_2. This can be performed in parallel, as well as the computation of the subdomain solutions in Step 4. This step can be considered as a correction of the right-hand sides ϕ_i, $i = 1, 2$, taking the interface solution into account.

4.3 Exact Decomposition

The main work is to solve the linear system $Sw_B = \tilde{\phi}_B$ in Step 3 in Algorithm 4.1. The capacitance matrix S is dense and maintains the properties of the matrix $\mathcal{A}_h$. Therefore S is positive definite, but not symmetric. There are several approaches for solving this system. The application of direct methods can be expensive. When using Krylov subspace methods, like GMRES or BiCGSTAB, each matrix vector product with S needs two subdomain solvers ($\mathcal{A}_1^{-1}$ and $\mathcal{A}_2^{-1}$), which can be performed in parallel, but this is still a lot of effort. One can show that the condition number of S is better than that one of $\mathcal{A}_h$, but still large with an order of $O(h^{-1})$ (instead of $O(h^{-2})$) and therefore a good preconditioner is useful, see [19, Section 1.2]. Another approach is a direct setup of the capacitance matrix. In [18], an exact eigendecomposition of the interface operator S is derived for the constant coefficient two-dimensional elliptic problem. In what follows, this procedure is generalized for the case of a coupled system of elliptic partial differential equations to be solved on a rectangle. An exact decomposition of the capacitance matrix S is presented, which can be efficiently computed and easily inverted.
Let Ω_h be a uniform mesh grid on Ω with grid size h in x_1-direction, i.e.

$$h = \frac{1}{n+1},$$

where n is the number of grid points in x_1-direction. Further assume that l_1 and l_2 are integral multiples of h with m_1 internal grid points in Ω_1 in x_2-direction and m_2 internal grid points in Ω_2, i.e.,

$$l_1 = (m_1 + 1)h \quad \text{and} \quad l_2 = (m_2 + 1)h$$

and that the interface B parallel to the x_1-axis, see Figure 4.3.

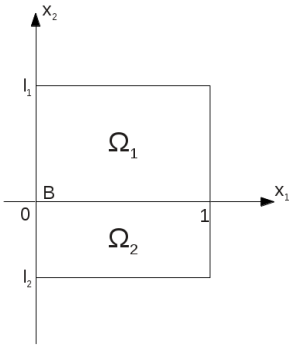


Figure 4.3: Two subdomains

Consider a finite difference discretization for the constant coefficient case where the uniformly elliptic operator L is approximated as described in Chapter 3 in a more general form

$$Lu = au(x_1 - h, x_2) + bu(x_1, x_2) + cu(x_1 + h, x_2) + du(x_1, x_2 - h) + eu(x_1, x_2 + h). \quad (4.11)$$

In stencil notation one has

$$\begin{bmatrix} & d & \\ a & b & c \\ & e & \end{bmatrix}_h$$

where it is assumed that a, c, d, e are nonzero.

THEOREM 4.3 (Exact decomposition of S)**.**
The Schur complement matrix S can be decomposed into

$$S = \bar{F}\Lambda\bar{F}^{-1}$$

where

$$\bar{F} = \begin{bmatrix} \tilde{F} & 0 \\ 0 & \tilde{F} \end{bmatrix}, \tilde{F} = DF$$

with

$$F_{ij} = \sqrt{2h}\sin\left(ij\pi h\right), i, j = 1, \ldots, n$$

and D a diagonal matrix, given by

$$D_{ii} = \left(\sqrt{\frac{a}{c}}\right)^i, i = 1, \ldots, n.$$

The decomposition of S given in Theorem 4.3 will be derived in the sequel. F is an orthogonal matrix with the property $F^T = F^{-1} = F$ and $\Lambda \in \mathbb{R}^{2n\times 2n}$ is a tri-blockdiagonal matrix of the form

$$\Lambda = \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{bmatrix}$$

where $\Lambda_{ij} \in \mathbb{R}^{n\times n}$, $i, j = 1, 2$ are diagonal matrices. Further define for $j = 1, \ldots, n$

$$w_j := \sqrt{2h}\left(\sin\left(j\pi h\right), \sin\left(2j\pi h\right), \ldots, \sin\left(nj\pi h\right)\right)^T \tag{4.12}$$

and

$$\tilde{w}_j := \sqrt{2h}\left(\left(\sqrt{\frac{a}{c}}\right)^1 \sin\left(j\pi h\right), \left(\sqrt{\frac{a}{c}}\right)^2 \sin\left(2j\pi h\right), \ldots, \left(\sqrt{\frac{a}{c}}\right)^n \sin\left(nj\pi h\right)\right)^T \tag{4.13}$$

as the columns of F and $\tilde{F}$, respectively.
The entries of the diagonal matrices Λ_{ij}, $i, j = 1, 2$, have to be determined. Therefore consider the products

$$S\begin{pmatrix} \tilde{w}_j \\ 0 \end{pmatrix} = \mathcal{A}_B\begin{pmatrix} \tilde{w}_j \\ 0 \end{pmatrix} - \mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\mathcal{A}_{1B}\begin{pmatrix} \tilde{w}_j \\ 0 \end{pmatrix} - \mathcal{A}_{2B}^T\mathcal{A}_2^{-1}\mathcal{A}_{2B}\begin{pmatrix} \tilde{w}_j \\ 0 \end{pmatrix}$$

and

$$S\begin{pmatrix} 0 \\ \tilde{w}_j \end{pmatrix} = \mathcal{A}_B\begin{pmatrix} 0 \\ \tilde{w}_j \end{pmatrix} - \mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\mathcal{A}_{1B}\begin{pmatrix} 0 \\ \tilde{w}_j \end{pmatrix} - \mathcal{A}_{2B}^T\mathcal{A}_2^{-1}\mathcal{A}_{2B}\begin{pmatrix} 0 \\ \tilde{w}_j \end{pmatrix}.$$

The derivation is splitted into three parts:

(1) $\mathcal{A}_B = \bar{F}\Lambda^B\bar{F}^{-1}$

(2) $-\mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\mathcal{A}_{1B} = \bar{F}\Lambda^1\bar{F}^{-1}$

(3) $-\mathcal{A}_{2B}^T\mathcal{A}_2^{-1}\mathcal{A}_{2B} = \bar{F}\Lambda^2\bar{F}^{-1}$.

Each of these parts has a contribution to the Schur complement which has to be quantified and then concatenated, i.e.

$$\begin{aligned} S &= \mathcal{A}_B - \mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\mathcal{A}_{1B} - \mathcal{A}_{2B}^T\mathcal{A}_2^{-1}\mathcal{A}_{2B} \\ &= \bar{F}\Lambda^B\bar{F}^{-1} + \bar{F}\Lambda^1\bar{F}^{-1} + \bar{F}\Lambda^2\bar{F}^{-1} \\ &= \bar{F}(\Lambda^B + \Lambda^1 + \Lambda^2)\bar{F}^{-1} \\ &= \bar{F}\Lambda\bar{F}. \end{aligned}$$

For simplifying some terms within the derivation the following preliminary lemma holds.

LEMMA 4.4.
Let $\sigma_j = 4\sin^2\left(\frac{j\pi h}{2}\right)$ and $\sin(ij\pi h) \neq 0$. Then

$$\frac{\sin((i-1)j\pi h) + \sin((i+1)j\pi h)}{\sin(ij\pi h)} = 2 - \sigma_j.$$

Proof. It holds that

$$\begin{aligned} &(1) \quad \sin x + \sin y = 2\sin\left(\frac{x+y}{2}\right)\cos\left(\frac{x-y}{2}\right) \\ &(2) \quad 2\cos x = 2 - 4\sin^2\left(\frac{x}{2}\right). \end{aligned}$$

Then

$$\begin{aligned} &\frac{\sin((i-1)j\pi h) + \sin((i+1)j\pi h)}{\sin(ij\pi h)} \\ &\overset{(1)}{=} \frac{2}{\sin(ij\pi h)}\sin\left(\frac{((i-1)-(i+1))j\pi h}{2}\right)\cos\left(\frac{((i-1)+(i+1))j\pi h}{2}\right) \\ &= \frac{2}{\sin(ij\pi h)}\sin(ij\pi h)\cos(j\pi h) \\ &= 2\cos(j\pi h) \\ &\overset{(2)}{=} 2 - 4\sin^2\left(\frac{j\pi h}{2}\right) \\ &= 2 - \sigma_j. \end{aligned}$$

□

4.3.1 Contribution of $\mathcal{A}_B$

Firstly, the contribution of $\mathcal{A}_B$ to the Schur complement is analyzed. Therefore, the terms

$$\mathcal{A}_B\begin{pmatrix}\tilde{w}_j\\0\end{pmatrix} \quad \text{and} \quad \mathcal{A}_B\begin{pmatrix}0\\\tilde{w}_j\end{pmatrix}$$

are considered. With the definition of the tridiagonal matrix

$$\tilde{L}_h := \text{tridiag}\{a, b, c\} \in \mathbb{R}^{n\times n},$$

where a, b, c are determined by the discretization (4.11), it holds for $j = 1, \ldots, n$

$$\begin{aligned} & \mathcal{A}_B \begin{pmatrix} \tilde{w}_j \\ 0 \end{pmatrix} = \Lambda^{B,j} \begin{pmatrix} \tilde{w}_j \\ 0 \end{pmatrix} \\ \Leftrightarrow \quad & \begin{bmatrix} \alpha\tilde{L}_h & -I_n \\ I_n & \tilde{L}_h \end{bmatrix} \begin{pmatrix} \tilde{w}_j \\ 0 \end{pmatrix} = \begin{bmatrix} \lambda_{11}^{B,j} & \lambda_{12}^{B,j} \\ \lambda_{21}^{B,j} & \lambda_{22}^{B,j} \end{bmatrix} \begin{pmatrix} \tilde{w}_j \\ 0 \end{pmatrix} \\ \Leftrightarrow \quad & \begin{pmatrix} \alpha\tilde{L}_h\tilde{w}_j \\ \tilde{w}_j \end{pmatrix} = \begin{pmatrix} \lambda_{11}^{B,j}\tilde{w}^j \\ \lambda_{21}^{B,j}\tilde{w}^j \end{pmatrix} \end{aligned}$$

and

$$\begin{aligned} & \mathcal{A}_B \begin{pmatrix} 0 \\ \tilde{w}_j \end{pmatrix} = \Lambda^{B,j} \begin{pmatrix} 0 \\ \tilde{w}_j \end{pmatrix} \\ \Leftrightarrow \quad & \begin{bmatrix} \alpha\tilde{L}_h & -I_n \\ I_n & \tilde{L}_h \end{bmatrix} \begin{pmatrix} 0 \\ \tilde{w}_j \end{pmatrix} = \begin{bmatrix} \lambda_{11}^{B,j} & \lambda_{12}^{B,j} \\ \lambda_{21}^{B,j} & \lambda_{22}^{B,j} \end{bmatrix} \begin{pmatrix} 0 \\ \tilde{w}_j \end{pmatrix} \\ \Leftrightarrow \quad & \begin{pmatrix} -\tilde{w}_j \\ \tilde{L}_h\tilde{w}_j \end{pmatrix} = \begin{pmatrix} \lambda_{12}^{B,j}\tilde{w}^j \\ \lambda_{22}^{B,j}\tilde{w}^j \end{pmatrix}. \end{aligned}$$

Now $\tilde{L}_h\tilde{w}_j$ has to be determined:

$$\begin{aligned} & (\tilde{L}_h\tilde{w}_j)_i \\ & = \sqrt{2h}\left(a\left(\sqrt{\frac{a}{c}}\right)^{i-1}\sin((i-1)j\pi h) + b\left(\sqrt{\frac{a}{c}}\right)^{i}\sin(ij\pi h) + c\left(\sqrt{\frac{a}{c}}\right)^{i+1}\sin((i+1)j\pi h)\right) \\ & = \sqrt{2h}\left(\sqrt{\frac{a}{c}}\right)^{i}\sin(ij\pi h)\left(\frac{a\left(\sqrt{\frac{a}{c}}\right)^{-1}\sin((i-1)j\pi h) + c\sqrt{\frac{a}{c}}\sin((i+1)j\pi h)}{\sin(ij\pi h)} + b\right) \\ & = \sqrt{2h}\left(\sqrt{\frac{a}{c}}\right)^{i}\sin(ij\pi h)\left(\sqrt{ac}\left(\frac{\sin((i-1)j\pi h) + \sin((i+1)j\pi h)}{\sin(ij\pi h)}\right) + b\right), i = 1, \ldots, n. \end{aligned}$$

Applying Lemma 4.4 for $\sigma_j = 4\sin^2(\frac{j\pi h}{2})$ and take (4.12) into account, this is equal to

$$= (b + \sqrt{ac}(2 - \sigma_j))\tilde{w}_j.$$

Therefore

$$\begin{aligned} \lambda_{11}^{B,j} &= \alpha(b + \sqrt{ac}(2 - \sigma_j)), \\ \lambda_{12}^{B,j} &= -1, \\ \lambda_{21}^{B,j} &= 1, \\ \lambda_{22}^{B,j} &= (b + \sqrt{ac}(2 - \sigma_j)), \end{aligned}$$

and the decomposition of $\mathcal{A}_B$ is given by

$$\mathcal{A}_B = \bar{F}\Lambda^B\bar{F}^{-1},$$

where

$$\Lambda^B = \begin{bmatrix} \Lambda^B_{11} & \Lambda^B_{12} \\ \Lambda^B_{21} & \Lambda^B_{22} \end{bmatrix} = \begin{bmatrix} \alpha\,\mathrm{diag}(b+\sqrt{ac}(2-\sigma_j)) & -I_n \\ I_n & \mathrm{diag}(b+\sqrt{ac}(2-\sigma_j)) \end{bmatrix}.$$

4.3.2 Contribution of $-\mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\mathcal{A}_{1B}$

Consider the second terms

$$-\mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\mathcal{A}_{1B}\begin{pmatrix}\tilde{w}_j\\0\end{pmatrix} \quad \text{and} \quad -\mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\mathcal{A}_{1B}\begin{pmatrix}0\\\tilde{w}_j\end{pmatrix}.$$

This can be computed by first solving the discrete equation on Ω_1 with homogeneous right-hand side and the boundary condition

$$w = \begin{pmatrix}\tilde{w}_j\\0\end{pmatrix} \quad \text{resp.} \quad w = \begin{pmatrix}0\\\tilde{w}_j\end{pmatrix}$$

on B and homogeneous Dirichlet boundary conditions elsewhere and then taking the solution on the first row of grid points above B multiplied by αe for y resp. e for p. Solve

$$\begin{aligned} \begin{bmatrix} \alpha L_h & -I_h \\ I_h & L_h \end{bmatrix}\begin{pmatrix}y\\p\end{pmatrix} &= \begin{pmatrix}0\\0\end{pmatrix} \quad \text{in } \Omega_1, \\ \begin{pmatrix}y\\p\end{pmatrix} &= \begin{pmatrix}0\\0\end{pmatrix} \quad \text{on } \partial\Omega_1 \setminus B \\ \begin{pmatrix}y\\p\end{pmatrix} &= \begin{pmatrix}\tilde{w}_j\\0\end{pmatrix} \quad \text{on } B \end{aligned} \tag{4.14}$$

resp.

$$\begin{aligned} \begin{bmatrix} \alpha L_h & -I_h \\ I_h & L_h \end{bmatrix}\begin{pmatrix}y\\p\end{pmatrix} &= \begin{pmatrix}0\\0\end{pmatrix} \quad \text{in } \Omega_2, \\ \begin{pmatrix}y\\p\end{pmatrix} &= \begin{pmatrix}0\\0\end{pmatrix} \quad \text{on } \partial\Omega_1 \setminus B \\ \begin{pmatrix}y\\p\end{pmatrix} &= \begin{pmatrix}0\\\tilde{w}_j\end{pmatrix} \quad \text{on } B. \end{aligned} \tag{4.15}$$

Consider the solution vectors $y(x_1, x_2)$, $p(x_1, x_2)$ of the form

$$\begin{aligned} y(ih, kh) &= d_k(\tilde{w}_j)_i = d_k\sqrt{2h}\left(\sqrt{\frac{a}{c}}\right)^i \sin(ij\pi h) \\ p(ih, kh) &= e_k(\tilde{w}_j)_i = e_k\sqrt{2h}\left(\sqrt{\frac{a}{c}}\right)^i \sin(ij\pi h), \end{aligned}$$

where $0 \leq i \leq n+1$ and $0 \leq k \leq m_1+1$, and substitute y into (4.14) with L_h discretized according to (4.11):

$$\begin{aligned}
0 =& \alpha \Bigg[a d_k \sqrt{2h} \left(\sqrt{\frac{a}{c}}\right)^{i-1} \sin((i-1)j\pi h) + b d_k \sqrt{2h} \left(\sqrt{\frac{a}{c}}\right)^{i} \sin(ij\pi h) \\
&+ c d_k \sqrt{2h} \left(\sqrt{\frac{a}{c}}\right)^{i+1} \sin((i+1)j\pi h) + d d_{k-1} \sqrt{2h} \left(\sqrt{\frac{a}{c}}\right)^{i} \sin(ij\pi h) \\
&+ e d_{k+1} \sqrt{2h} \left(\sqrt{\frac{a}{c}}\right)^{i} \sin(ij\pi h) \Bigg] - e_k \sqrt{2h} \left(\sqrt{\frac{a}{c}}\right)^{i} \sin(ij\pi h) \\
=& \Bigg[\alpha \Big(\frac{a d_k \left(\sqrt{\frac{a}{c}}\right)^{-1} \sin((i-1)j\pi h)}{\sin(ij\pi h)} + b d_k + \frac{c d_k \sqrt{\frac{a}{c}} \sin((i+1)j\pi h)}{\sin(ij\pi h)} \\
&+ d d_{k-1} + e d_{k+1} \Big) - e_k \Bigg] \sqrt{2h} \left(\sqrt{\frac{a}{c}}\right)^{i} \sin(ij\pi h) \\
=& \alpha \left[b + \sqrt{ac} \Big(\frac{\sin((i-1)j\pi h) + \sin((i+1)j\pi h)}{\sin(ij\pi h)} \Big) d_k + d d_{k-1} + e d_{k+1} \right] - e_k
\end{aligned}$$

Owing to Lemma 4.4 it holds

$$\alpha \left[(b + \sqrt{ac}(2 - \sigma_j)) d_k + d d_{k-1} + e d_{k+1} \right] - e_k = 0.$$

An analogue analysis for p proves

$$\left[e e_{k+1} + (b + \sqrt{ac}(2 - \sigma_j)) e_k + d e_{k-1} \right] + d_k = 0.$$

This is a system of second order linear difference equations. To fulfill the boundary condition in (4.14) and (4.15), the conditions

$$\begin{aligned}
d_0 = 1, e_0 = 0, d_{m_1+1} = 0, e_{m_1+1} = 0 \text{ for (4.14)} \\
\text{resp. } d_0 = 0, e_0 = 1, d_{m_1+1} = 0, e_{m_1+1} = 0 \text{ for (4.15)}
\end{aligned}$$

have to be enforced. Summarized, the determination of

$$\Lambda^1 = \begin{bmatrix} \Lambda^1_{11} & \Lambda^1_{12} \\ \Lambda^1_{22} & \Lambda^1_{22} \end{bmatrix}$$

is reduced to solving the systems

$$\begin{aligned}
\alpha \left[(b + \sqrt{ac}(2 - \sigma_j)) \bar{d}_k + d \bar{d}_{k-1} + e \bar{d}_{k+1} \right] - \bar{e}_k = 0 \\
\left[e \bar{e}_{k+1} + (b + \sqrt{ac}(2 - \sigma_j)) \bar{e}_k + d \bar{e}_{k-1} \right] + \bar{d}_k = 0
\end{aligned} \tag{4.16}$$

with the boundary conditions

$$\bar{d}_0 = 1, \bar{e}_0 = 0, \bar{d}_{m_1+1} = 0, \bar{e}_{m_1+1} = 0$$

and

$$\begin{aligned}
\alpha \left[(b + \sqrt{ac}(2 - \sigma_j)) \tilde{d}_k + d \tilde{d}_{k-1} + e \tilde{d}_{k+1} \right] - \tilde{e}_k = 0 \\
\left[e \tilde{e}_{k+1} + (b + \sqrt{ac}(2 - \sigma_j)) \tilde{e}_k + d \tilde{e}_{k-1} \right] + \tilde{d}_k = 0
\end{aligned} \tag{4.17}$$

with the boundary conditions

$$\tilde{d}_0 = 0, \tilde{e}_0 = 1, \tilde{d}_{m_1+1} = 0, \tilde{e}_{m_1+1} = 0$$

for $j = 1, \ldots, n$ and taking the solution of the first row of grid points above B, i.e. $\bar{d}_1, \tilde{d}_1$ and $\bar{e}_1, \tilde{e}_1$ multiplied by αe resp. e. Let $\bar{d}^j, \tilde{d}^j$ and $\bar{e}^j, \tilde{e}^j$ denote the solution of the j-th problem. This results in

$$\Lambda^1 = \begin{bmatrix} \alpha e \operatorname{diag}(\bar{d}_1^j) & \alpha e \operatorname{diag}(\tilde{d}_1^j) \\ e \operatorname{diag}(\bar{e}_1^j) & e \operatorname{diag}(\tilde{e}_1^j) \end{bmatrix}.$$

To save some efforts a similarity between the structure and the solution of the two systems (4.16) and (4.17) can be exploited. Written as a linear system with the boundary conditions on the right-hand side, omit the index j and define

$$\begin{aligned} D &:= \operatorname{tridiag}\{e, b + \sqrt{ac}(2 - \sigma_j), d\} \in \mathbb{R}^{m_1 \times m_1}, \\ I_{m_1} &\in \mathbb{R}^{m_1 \times m_1} \text{ identity matrix }, 0 \in \mathbb{R}^{m_1} \\ \text{and } g &:= [d, 0, \ldots, 0]^T \in \mathbb{R}^{m_1}. \end{aligned}$$

Hence the linear systems in (4.16) take the form

$$\begin{bmatrix} \alpha D & -I_{m_1} \\ I_{m_1} & D \end{bmatrix} \begin{pmatrix} \bar{d} \\ \bar{e} \end{pmatrix} = \begin{pmatrix} \alpha g \\ 0 \end{pmatrix}.$$

A reformulation of the system leads to

$$\begin{aligned} & \begin{bmatrix} \alpha D & -I_{m_1} \\ I_{m_1} & D \end{bmatrix} \begin{pmatrix} \bar{d} \\ \bar{e} \end{pmatrix} = \begin{pmatrix} \alpha g \\ 0 \end{pmatrix} \\ \Leftrightarrow & \begin{bmatrix} D & I_{m_1} \\ -I_{m_1} & \alpha D \end{bmatrix} \begin{pmatrix} \bar{e} \\ \bar{d} \end{pmatrix} = \begin{pmatrix} 0 \\ \alpha g \end{pmatrix} \end{aligned}$$

and setting $\bar{e} := -\alpha \tilde{d}$ and $\bar{d} := \tilde{e}$, this is equivalent to

$$\begin{bmatrix} \alpha D & -I_{m_1} \\ I_{m_1} & D \end{bmatrix} \begin{pmatrix} \tilde{d} \\ \tilde{e} \end{pmatrix} = \begin{pmatrix} 0 \\ g \end{pmatrix}$$

which is the linear system for the problem (4.17). So $\bar{d} = \tilde{e}$ and $\bar{e} = -\alpha \tilde{d}$, and in particular $\bar{d}_1 = \tilde{e}_1$ and $\bar{e}_1 = -\alpha \tilde{d}_1$. Therefore it is sufficient to solve (4.16) and the resulting decomposition is given by

$$-\mathcal{A}_{1B} \mathcal{A}_1^{-1} \mathcal{A}_{1B} = \bar{F} \Lambda^1 \bar{F}^{-1}$$

where

$$\Lambda^1 = \begin{bmatrix} \Lambda_{11}^1 & \Lambda_{12}^1 \\ \Lambda_{21}^1 & \Lambda_{22}^1 \end{bmatrix} = \begin{bmatrix} \alpha e \operatorname{diag}(\bar{d}_1^j) & -e \operatorname{diag}(\bar{e}_1^j) \\ e \operatorname{diag}(\bar{e}_1^j) & e \operatorname{diag}(\bar{d}_1^j) \end{bmatrix}.$$

REMARK 4.5 (Solution of the linear systems)**.**
The main approach for solving systems of homogeneous second order difference equations (also: *recurrence equations*) is given in Algorithm 4.2.

Algorithm 4.2 General approach for solving second order linear difference equation

For $j = 1, \ldots, n$ do

1: Reformulate the system as a first order system $y(k+1) = My(k)$:

$$\begin{pmatrix} y_1(k+1) \\ y_2(k+1) \\ y_3(k+1) \\ y_4(k+1) \end{pmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{d}{e} & \frac{-\eta}{e} & 0 & \frac{1}{\alpha e} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{1}{e} & -\frac{d}{e} & -\frac{\eta}{e} \end{bmatrix} \begin{pmatrix} y_1(k) \\ y_2(k) \\ y_3(k) \\ y_4(k) \end{pmatrix}$$

where $\eta = b + \sqrt{ac}(2 - \sigma_j)$.

2: Compute eigenvalues λ_i, $i = 1, \ldots, 4$, and the corresponding eigenvectors ξ_i to get a general solution of this system

$$y(k) = c_1\lambda_1^k\xi_1 + \cdots + c_4\lambda_4^k\xi_4.$$

3: To obtain the special solution of the difference equations, the boundaries are imposed:

$$\begin{aligned} &y_2(0) = 1, && y_2(m_1+1) = 0 \\ &y_4(0) = 0, && y_4(m_1+1) = 0 \quad \text{for (4.16)} \end{aligned}$$

and

$$\begin{aligned} &y_2(0) = 0, && y_2(m_1+1) = 0 \\ &y_4(0) = 1, && y_4(m_1+1) = 0 \quad \text{for (4.17)} \end{aligned}$$

and the resulting linear system

$$\begin{bmatrix} \xi_{21} & \xi_{22} & \xi_{23} & \xi_{24} \\ \lambda_1^{m1+1}\xi_{21} & \lambda_2^{m1+1}\xi_{22} & \lambda_3^{m1+1}\xi_{23} & \lambda_4^{m1+1}\xi_{24} \\ \xi_{41} & \xi_{42} & \xi_{43} & \xi_{44} \\ \lambda_1^{m1+1}\xi_{41} & \lambda_2^{m1+1}\xi_{42} & \lambda_3^{m1+1}\xi_{43} & \lambda_4^{m1+1}\xi_{44} \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ resp. } \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{4.18}$$

is solved for the desired values

$$\begin{aligned} d_1 &= c_1\lambda_1\xi_{21} + c_2\lambda_2\xi_{22} + c_3\lambda_3\xi_{23} + c_4\lambda_4\xi_{24} \\ e_1 &= c_1\lambda_1\xi_{41} + c_2\lambda_2\xi_{42} + c_3\lambda_3\xi_{43} + c_4\lambda_4\xi_{44}. \end{aligned}$$

A problem arises when solving the linear system (4.18): if m_1 is very large, i.e. the case of large domains or for very fine discretizations, the entries $\lambda_i^{m_1+1}$ for the eigenvalues λ_i in the range $(-1, 1)$ are numerically treated as zero and the eigenvalues beyond these as INF. For that reason this algorithm is not applied in this thesis.
As the resulting system of second order difference equation has a similar structure to a discretized one-dimensional general differential operator, this system can be solved efficiently using a multigrid method.

4.3.3 Contribution of $-\mathcal{A}_{2B}^T \mathcal{A}_2^{-1} \mathcal{A}_{2B}$

Consider the third terms

$$-\mathcal{A}_{2B}^T \mathcal{A}_2^{-1} \mathcal{A}_{2B} \begin{pmatrix} \tilde{w}_j \\ 0 \end{pmatrix} \quad \text{and} \quad -\mathcal{A}_{2B}^T \mathcal{A}_2^{-1} \mathcal{A}_{2B} \begin{pmatrix} 0 \\ \tilde{w}_j \end{pmatrix}.$$

As for the second term, this can be computed analogously by first solving the discrete equation on Ω_2 with homogeneous right-hand side and the boundary condition

$$w = \begin{pmatrix} \tilde{w}_j \\ 0 \end{pmatrix} \quad \text{resp.} \quad w = \begin{pmatrix} 0 \\ \tilde{w}_j \end{pmatrix}$$

on B and homogeneous boundary conditions elsewhere, and then taking the solution in the first row of the grid points below B multiplied by αd for y resp. d for p. Let $\hat{d}^j$, $\mathring{d}^j$ and $\hat{e}^j$, $\mathring{e}^j$ denote the solution of the j-th problem. As in the proceeding section, the following systems has to be solved

$$\begin{aligned} \alpha \left[(b + \sqrt{ac}(2 - \sigma_j)) \mathring{d}_k + d \mathring{d}_{k-1} + e \mathring{d}_{k+1} \right] - \mathring{e}_k &= 0 \\ \left[e \mathring{e}_{k+1} + (b + \sqrt{ac}(2 - \sigma_j)) \mathring{e}_k + d \mathring{e}_{k-1} \right] + \mathring{d}_k &= 0 \end{aligned} \tag{4.19}$$

with the boundary conditions

$$\mathring{d}_0 = 1, \mathring{e}_0 = 0, \mathring{d}_{m_2+1} = 0, \mathring{e}_{m_2+1} = 0$$

and

$$\begin{aligned} \alpha \left[(b + \sqrt{ac}(2 - \sigma_j)) \hat{d}_k + d \hat{d}_{k-1} + e \hat{d}_{k+1} \right] - \hat{e}_k &= 0 \\ \left[e \hat{e}_{k+1} + (b + \sqrt{ac}(2 - \sigma_j)) \hat{e}_k + d \hat{e}_{k-1} \right] + \hat{d}_k &= 0 \end{aligned} \tag{4.20}$$

with the boundary conditions

$$\hat{d}_0 = 0, \hat{e}_0 = 1, \hat{d}_{m_2+1} = 0, \hat{e}_{m_2+1} = 0$$

for $j = 1, \ldots, n$ and taking the solution of the first row of grid points above B, i.e. $\mathring{d}_1, \hat{d}_1$ and $\mathring{e}_1, \hat{e}_1$ multiplied by αd resp. d. Let $\mathring{d}_1^j = \hat{e}_1^j$ and $\mathring{e}_1^j = -\alpha \hat{d}_1^j$ denote the solution of the j-th problem this ends in the resulting decomposition

$$-\mathcal{A}_{2B} \mathcal{A}_2^{-1} \mathcal{A}_{2B} = \bar{F} \Lambda^2 \bar{F}^{-1}$$

where

$$\Lambda^2 = \begin{bmatrix} \Lambda_{11}^2 & \Lambda_{12}^2 \\ \Lambda_{21}^2 & \Lambda_{22}^2 \end{bmatrix} = \begin{bmatrix} \alpha d \operatorname{diag}(\mathring{d}_1^j) & -d \operatorname{diag}(\mathring{e}_1^j) \\ d \operatorname{diag}(\mathring{e}_1^j) & d \operatorname{diag}(\mathring{d}_1^j) \end{bmatrix}.$$

4.3.4 Summary and Remarks

Summarized it holds

$$S = \bar{F}\Lambda\bar{F}^{-1}$$

with

$$\begin{aligned}
\Lambda &= \Lambda^B + \Lambda^1 + \Lambda^2 \\
&= \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{bmatrix} \\
&= \begin{bmatrix} \Lambda^B_{11} + \Lambda^1_{11} + \Lambda^2_{11} & \Lambda^B_{12} + \Lambda^1_{12} + \Lambda^2_{12} \\ \Lambda^B_{21} + \Lambda^1_{21} + \Lambda^2_{21} & \Lambda^B_{22} + \Lambda^1_{22} + \Lambda^2_{22} \end{bmatrix} \\
&= \begin{bmatrix} \alpha\,\mathrm{diag}(b + \sqrt{ac}(2-\sigma_j) + e\bar{d}^j_1 + d\mathring{d}^j_1) & -\mathrm{diag}(1 + e\bar{e}^j_1 + d\mathring{e}^j_1) \\ \mathrm{diag}(1 + e\bar{e}^j_1 + d\mathring{e}^j_1) & \mathrm{diag}(b + \sqrt{ac}(2-\sigma_j) + e\bar{d}^j_1 + d\mathring{d}^j_1) \end{bmatrix}
\end{aligned}$$

and with the obvious fact that $\Lambda_{11} = \alpha\Lambda_{22}$ and $\Lambda_{12} = -\Lambda_{21}$ the matrix Λ has the representation

$$\Lambda = \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ -\Lambda_{12} & \frac{1}{\alpha}\Lambda_{11} \end{bmatrix}.$$

REMARK 4.6.

1. The Schur complement only depends on the number n of grid points on the interface and the numbers m_1 and m_2 of internal grid points in y-direction (and, of course, on the discretization of the differential operators). With tribute to this fact, write $S = S(n, m_1, m_2)$ if necessary to point out this dependency.
2. It is worth mentioning to be careful with the nomenclature: whereas e and d stems from the discretization of the differential operator in (4.11), $\bar{e}_1$ and $\bar{d}_1$ denotes the solution of a coupled difference equation and taking the value of the first grid point.

It is suitable to clarify the computation by means of an example.

EXAMPLE 4.7.
Consider the optimality system $\mathcal{A}$ for the academic optimal control problem with the Laplace operator $L = -\Delta$. Let $\Omega = (0,1)^2$ be decomposed into $\Omega_1 = (0,1) \times (0, 0.5)$, $\Omega_2 = (0,1) \times (0.5, 1)$ and $B = (0,1) \times 0.5$. The difference stencil is given by

$$-\frac{1}{h^2}\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}_h.$$

As $\Omega_1 = \Omega_2$, $m_1 = m_2$ and as the Schur complement only depends on the size of the subdomains, $\bar{d}^j_1 = \mathring{d}^j_1 (:= d^j_1)$ and $\bar{e}^j_1 = \mathring{e}^j_1 (:= e^j_1)$, the matrix Λ has the form

$$\Lambda = \begin{bmatrix} -\frac{\alpha}{h^2}\,\mathrm{diag}(2d^j_1 - (2+\sigma_j)) & -\frac{1}{h^2}\,\mathrm{diag}(2e^j_1 - h^2) \\ \frac{1}{h^2}\,\mathrm{diag}(2e^j_1 - h^2) & -\frac{1}{h^2}\,\mathrm{diag}(2d^j_1 - (2+\sigma_j)) \end{bmatrix}$$

and $D = \operatorname{diag}\left(\left(\sqrt{\frac{a}{c}}\right)^i\right), i = 1, \ldots, n$ simplifies to the identity matrix I_n. The resulting Schur complement matrix is given by

$$S = \begin{bmatrix} F & 0 \\ 0 & F \end{bmatrix} \begin{bmatrix} -\frac{\alpha}{h^2}\operatorname{diag}(2d_1^j - (2+\sigma_j)) & -\frac{1}{h^2}\operatorname{diag}(2e_1^j - h^2) \\ \frac{1}{h^2}\operatorname{diag}(2e_1^j - h^2) & -\frac{1}{h^2}\operatorname{diag}(2d_1^j - (2+\sigma_j)) \end{bmatrix} \begin{bmatrix} F & 0 \\ 0 & F \end{bmatrix}.$$

The algorithm for computing the Schur complement for the optimality system is given below in Algorithm 4.3.

Algorithm 4.3 Schur complement algorithm

Let $Lu = au(x_1 - h, x_2) + bu(x_1, x_2) + cu(x_1 + h, x_2) + du(x_1, x_2 - h) + eu(x_1, x_2 + h)$ and Ω decomposed in two strips Ω_1 and Ω_2, where m_1 resp. m_2 are the number of internal grid points in x_2-direction and n is the numbers of internal grid points in x_1-direction.

1: For $j = 1, \ldots, n$ solve:

$$\begin{aligned} \alpha\left[(b + \sqrt{ac}(2-\sigma_j))\bar{d}_k + d\bar{d}_{k-1} + e\bar{d}_{k+1}\right] - \bar{e}_k &= 0 \\ \left[e\bar{e}_{k+1} + (b + \sqrt{ac}(2-\sigma_j))\bar{e}_k + d\bar{e}_{k-1}\right] + \bar{d}_k &= 0 \end{aligned}$$

with the boundary conditions

$$\bar{d}_0 = 1, \bar{e}_0 = 0, \bar{d}_{m_1+1} = 0, \bar{e}_{m_1+1} = 0$$

and take the values of the first grid point $\bar{d}_1$ and $\bar{e}_1$.

2: For $j = 1, \ldots, n$ solve:

$$\begin{aligned} \alpha\left[(b + \sqrt{ac}(2-\sigma_j))\mathring{d}_k + d\mathring{d}_{k-1} + e\mathring{d}_{k+1}\right] - \mathring{e}_k &= 0 \\ \left[e\mathring{e}_{k+1} + (b + \sqrt{ac}(2-\sigma_j))\mathring{e}_k + d\mathring{e}_{k-1}\right] + \mathring{d}_k &= 0 \end{aligned}$$

with the boundary conditions

$$\mathring{d}_0 = 1, \mathring{e}_0 = 0, \mathring{d}_{m_2+1} = 0, \mathring{e}_{m_2+1} = 0$$

and take the values of the first grid point $\mathring{d}_1$ and $\mathring{e}_1$.

3: Set up Λ:

$$\begin{aligned} \Lambda &= \Lambda^1 + \Lambda^2 + \Lambda^3 = \\ &= \begin{bmatrix} \alpha\operatorname{diag}(b + \sqrt{ac}(2-\sigma_j) + e\bar{d}_1^j + d\mathring{d}_1^j) & \operatorname{diag}(1 - e\bar{e}_1^j - d\mathring{e}_1^j) \\ -\operatorname{diag}(1 - e\bar{e}_1^j - d\mathring{e}_1^j) & \operatorname{diag}(b + \sqrt{ac}(2-\sigma_j) + e\bar{d}_1^j + d\mathring{d}_1^j) \end{bmatrix}. \end{aligned}$$

4: Compute:

$$S = F\Lambda F^{-1}.$$

4.4 Computing the Inverse of S

For solving the linear system $Sw_B = \tilde{\phi}_B$ for the unknowns on the interface B, the inverse S^{-1} of S is required. The following preliminary remark states that the inversion of a general tri-blockdiagonal matrix can efficiently be done by block elimination.

REMARK 4.8 (Inverting a tri-blockdiagonal matrix)**.**
The inverse of a tri-blockdiagonal matrix $M \in \mathbb{R}^{2n\times 2n}$ of the form

$$M := \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} \operatorname{diag}(a_j) & \operatorname{diag}(b_j) \\ \operatorname{diag}(c_j) & \operatorname{diag}(d_j) \end{bmatrix}, j = 1, \ldots, n,$$

(where $AD - BC$ is assumed to be invertible) can be computed by

$$\begin{aligned} M^{-1} &= \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \\ &= \begin{bmatrix} (AD-BC)^{-1}D & -(AD-BC)^{-1}B \\ -(AD-BC)^{-1}C & (AD-BC)^{-1}A \end{bmatrix} \\ &= \begin{bmatrix} \operatorname{diag}(\frac{d_j}{a_jd_j-c_jb_j}) & -\operatorname{diag}(\frac{b_j}{a_jd_j-c_jb_j}) \\ -\operatorname{diag}(\frac{c_j}{a_jd_j-c_jb_j}) & \operatorname{diag}(\frac{a_j}{a_jd_j-c_jb_j}) \end{bmatrix}. \end{aligned}$$

With this remark, the inverse S^{-1} of $S = \bar{F}\Lambda\bar{F}^{-1}$ is determined by

$$\begin{aligned} S^{-1} &= (\bar{F}\Lambda\bar{F}^{-1})^{-1} \\ &= \bar{F}\Lambda^{-1}\bar{F}^{-1} \\ &= \begin{bmatrix} \tilde{F} & 0 \\ 0 & \tilde{F} \end{bmatrix} \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{bmatrix}^{-1} \begin{bmatrix} \tilde{F} & 0 \\ 0 & \tilde{F} \end{bmatrix}^{-1} \\ &= \begin{bmatrix} DF & 0 \\ 0 & DF \end{bmatrix} \begin{bmatrix} \Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22} \end{bmatrix}^{-1} \begin{bmatrix} FD^{-1} & 0 \\ 0 & FD^{-1} \end{bmatrix}, \end{aligned}$$

and applying Remark 4.8, this is equal to

$$\begin{aligned} &\begin{bmatrix} DF & 0 \\ 0 & DF \end{bmatrix} \begin{bmatrix} (\Lambda_{11}\Lambda_{22}-\Lambda_{21}\Lambda_{12})^{-1}\Lambda_{22} & -(\Lambda_{11}\Lambda_{22}-\Lambda_{21}\Lambda_{12})^{-1}\Lambda_{12} \\ -(\Lambda_{11}\Lambda_{22}-\Lambda_{21}\Lambda_{12})^{-1}\Lambda_{21} & (\Lambda_{11}\Lambda_{22}-\Lambda_{21}\Lambda_{12})^{-1}\Lambda_{11} \end{bmatrix} \begin{bmatrix} FD^{-1} & 0 \\ 0 & FD^{-1} \end{bmatrix} \\ &= \begin{bmatrix} DF(\Lambda_{11}\Lambda_{22}-\Lambda_{21}\Lambda_{12})^{-1}\Lambda_{22}FD^{-1} & -DF(\Lambda_{11}\Lambda_{22}-\Lambda_{21}\Lambda_{12})^{-1}\Lambda_{12}FD^{-1} \\ -DF(\Lambda_{11}\Lambda_{22}-\Lambda_{21}\Lambda_{12})^{-1}\Lambda_{21}FD^{-1} & DF(\Lambda_{11}\Lambda_{22}-\Lambda_{21}\Lambda_{12})^{-1}\Lambda_{11}FD^{-1} \end{bmatrix} \end{aligned}$$

and as stated in the preceding section, inserting $\Lambda_{11} = \alpha\Lambda_{22}$ and $\Lambda_{12} = -\Lambda_{21}$ leads to

$$\begin{aligned} &= \begin{bmatrix} \frac{1}{\alpha}DF(\frac{1}{\alpha}\Lambda_{11}^2+\Lambda_{12}^2)^{-1}\Lambda_{11}FD^{-1} & -DF(\frac{1}{\alpha}\Lambda_{11}^2+\Lambda_{12}^2)^{-1}\Lambda_{12}FD^{-1} \\ DF(\frac{1}{\alpha}\Lambda_{11}^2+\Lambda_{12}^2)^{-1}\Lambda_{12}FD^{-1} & DF(\frac{1}{\alpha}\Lambda_{11}^2+\Lambda_{12}^2)^{-1}\Lambda_{11}FD^{-1} \end{bmatrix} \\ &= \begin{bmatrix} \bar{S}_{11} & \bar{S}_{12} \\ \bar{S}_{21} & \bar{S}_{22} \end{bmatrix}. \end{aligned}$$

As $\alpha\bar{S}_{11} = \bar{S}_{22}$ and $\bar{S}_{12} = -\bar{S}_{21}$, it is sufficient to compute

$$\bar{S}_{22} = DF(\frac{1}{\alpha}\Lambda_{11}^2+\Lambda_{12}^2)^{-1}\Lambda_{11}FD^{-1} \text{ and } \bar{S}_{21} = DF(\frac{1}{\alpha}\Lambda_{11}^2+\Lambda_{12}^2)^{-1}\Lambda_{12}FD^{-1}.$$

If the explicit setup of the matrix S^{-1} is not wanted and only the action on a particular vector w_B is of interest, the entries of $\bar{S}_{21}$ and $\bar{S}_{22}$ can be computed by the following formulas. The entries of the matrices $\bar{S}_{22}$ and $\bar{S}_{21}$ are given by

$$(\bar{S}_{22})_{ij} = \frac{2h}{\alpha}\left(\sqrt{\frac{a}{c}}\right)^{i-j} \sum_{k=1}^{n} \frac{(\lambda_{11})_k}{\frac{1}{\alpha}(\lambda_{11})_k^2 + (\lambda_{12})_k^2} \sin(ik\pi h)\sin(jk\pi h)$$

and

$$(\bar{S}_{21})_{ij} = \frac{2h}{\alpha}\left(\sqrt{\frac{a}{c}}\right)^{i-j} \sum_{k=1}^{n} \frac{(\lambda_{12})_k}{\frac{1}{\alpha}(\lambda_{12})_k^2 + (\lambda_{12})_k^2} \sin(ik\pi h)\sin(jk\pi h)$$

for $i, j = 1, \ldots, n$.

Therewith the solution w_B of the linear system $Sw_B = \tilde{\phi}_B$ can be computed by

$$\begin{aligned} & S^{-1}\tilde{\phi}_B = w_B \\ \Leftrightarrow & \begin{bmatrix} \bar{S}_{11} & \bar{S}_{12} \\ \bar{S}_{21} & \bar{S}_{22} \end{bmatrix} \begin{pmatrix} \tilde{\phi}_y \\ \tilde{\phi}_p \end{pmatrix} = \begin{pmatrix} y_B \\ p_B \end{pmatrix} \\ \Leftrightarrow & \begin{bmatrix} \bar{S}_{11}\tilde{\phi}_y + \bar{S}_{12}\tilde{\phi}_p \\ \bar{S}_{21}\tilde{\phi}_y + \bar{S}_{22}\tilde{\phi}_p \end{bmatrix} = \begin{pmatrix} y_B \\ p_B \end{pmatrix} \\ \Leftrightarrow & \begin{bmatrix} \frac{1}{\alpha}\bar{S}_{22}\tilde{\phi}_y - \bar{S}_{21}\tilde{\phi}_p \\ \bar{S}_{21}\tilde{\phi}_y + \bar{S}_{22}\tilde{\phi}_p \end{bmatrix} = \begin{pmatrix} y_B \\ p_B \end{pmatrix} \end{aligned}$$

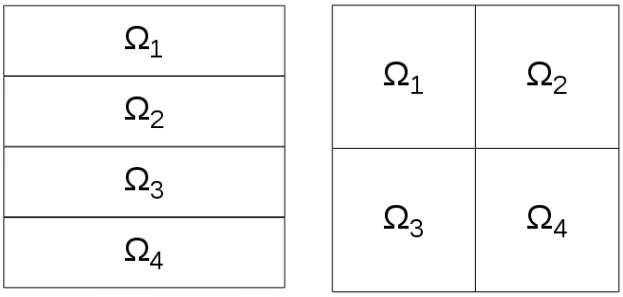
and given in a procedural formulation

$$(w_B)_i = \begin{pmatrix} y_B \\ p_B \end{pmatrix}_i = \begin{pmatrix} \sum_{k=1}^{n} \frac{1}{\alpha}(\bar{S}_{22})_{ik}(\tilde{\phi}_y)_k - (\bar{S}_{21})_{ik}(\tilde{\phi}_p)_k \\ \sum_{k=1}^{n} (\bar{S}_{21})_{ik}(\tilde{\phi}_y)_k + (\bar{S}_{22})_{ik}(\tilde{\phi}_p)_k \end{pmatrix}$$

for $i = 1, \ldots, n$.

4.5 Multi-Subdomain Decomposition

In this section the nonoverlapping domain decomposition for the case of a partition into many subdomains is considered. Here the interface solver needs to allow for more complex geometries of the interface and to take account of the stronger global coupling between various subdomains. As there are more subdomains, the interfaces lie closer together. Local information between adjacent subdomains is shared by the Schur complement, global coupling between distant subdomains is provided by the two proposed algorithms, the recursive solver or the simultaneous solver. There are two straightforward possibilities to divide a rectangular domain into more than two regions: *horizontal strips* or *boxes*, see Figure 4.4.
When dividing in boxes, the problem of so-called *cross-points* arise, which are more difficult to handle, see [10]. By dividing into strips, the ratio of the amount of communication and



(a) horizontal strips (b) boxes

Figure 4.4: Multi-domain decomposition

the amount of computation required grows as the strips become narrower. In this thesis, the decomposition into horizontal strips is considered.
The nonoverlapping domain decomposition into two regions can be extended to the case of many subdomains upright. Let Ω be partitioned into k nonoverlapping subregions with interface B,

$$\Omega = \Omega_1 \cup \Omega_2 \cup \cdots \cup \Omega_k \cup B \quad \text{where } \Omega_i \cap \Omega_j = \varnothing \text{ for } i \neq j.$$

The interface B is given by

$$B = \left\{ \bigcup_{i=1}^{k} \partial\Omega_i \right\} \cap \Omega,$$

and let $B_{i,i+1}$, $i = 1, \ldots, k-1$ be the part of the interface shared by the subdomains Ω_i and Ω_{i+1}. Let $I = \bigcup_{i=1}^{k} I_i$ denote the indices of the nodes lying in the interior of the subdomains and let I_B consist of the nodes on the interfaces B. According to this, the linear system $\mathcal{A}_h w_h = \phi_h$ can be reordered and written in block form

$$\begin{bmatrix} \mathcal{A}_I & \mathcal{A}_{IB} \\ \mathcal{A}_{IB}^T & \mathcal{A}_B \end{bmatrix} \begin{pmatrix} w_I \\ w_B \end{pmatrix} = \begin{pmatrix} \phi_I \\ \phi_B \end{pmatrix}.$$

Since a five point finite difference discretization is used, $\mathcal{A}_I$ is block-diagonal

$$\mathcal{A}_I = \text{blockdiag}(\mathcal{A}_i) = \begin{bmatrix} \mathcal{A}_1 & & 0 \\ & \ddots & \\ 0 & & \mathcal{A}_k \end{bmatrix}.$$

and as there is no coupling between points on two different interfaces, $\mathcal{A}_B$ is block-diagonal and the submatrix $\mathcal{A}_{IB}$ has the following form

$$\mathcal{A}_{IB} = \begin{bmatrix} \mathcal{A}_{1B_{12}} & & & & \\ \mathcal{A}_{2B_{12}} & \mathcal{A}_{2B_{23}} & & & \\ & \ddots & \ddots & & \\ & & & \mathcal{A}_{k-1B_{k-2,k-1}} & \mathcal{A}_{k-1B_{k-1,k}} \\ & & & & \mathcal{A}_{kB_{k-1,k}} \end{bmatrix}$$

where $\mathcal{A}_{iB_{j,j+1}}$ corresponds to the coupling between the unknowns inside the subdomain Ω_i with the unknowns on the interface $B_{j,j+1}$. As in the case of the two subdomains, the Schur

complement S is defined by

$$S = \mathcal{A}_B - \mathcal{A}_{IB}^T \mathcal{A}_I^{-1} \mathcal{A}_{IB}$$

and $\tilde{\phi}_B = \phi_B - \mathcal{A}_{IB}^T \mathcal{A}_I^{-1} \phi_I$. The Schur complement for the multiple subdomain case has similar properties to the two subdomain case, see [19, Section 3.2]. There are at least two different possibilities to solve the Schur complement system: a *recursive approach* and an *simultaneous approach*, introduced in the subsequent Sections 4.5.1 and 4.5.2.

4.5.1 Recursive Solver

The idea of the recursive approach, also called *nested dissection* is as follows: The domain Ω is divided into two strips Ω_1 and Ω_2. This partition is called P_1 and for a given partition P_i, P_{i+1} is obtained by subdividing each subdomain of P_i into two horizontal strips. This subdivision can also be done in two vertical strips, which allows to handle cross points.
The partition P_1 and the resulting linear system

$$\begin{bmatrix} \mathcal{A}_1 & 0 & \mathcal{A}_{1B_{1,2}} \\ 0 & \mathcal{A}_2 & \mathcal{A}_{2B_{1,2}} \\ \mathcal{A}_{1B_{1,2}}^T & \mathcal{A}_{2B_{1,2}}^T & \mathcal{A}_{B_{1,2}} \end{bmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_B \end{pmatrix} = \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_B \end{pmatrix}$$

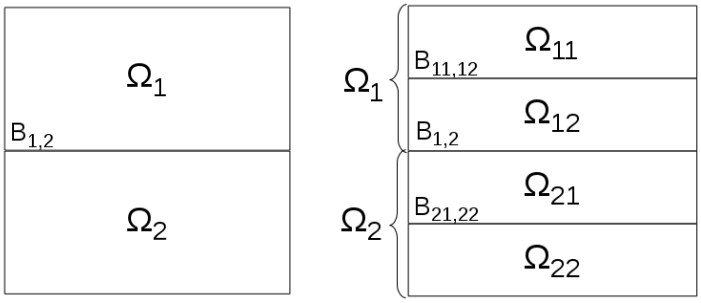
is further decomposed into

$$\begin{bmatrix} \mathcal{A}_{11} & 0 & \mathcal{A}_{1B_{11,12}} \\ 0 & \mathcal{A}_{12} & \mathcal{A}_{2B_{11,12}} \\ \mathcal{A}_{1B_{11,12}}^T & \mathcal{A}_{2B_{11,12}}^T & \mathcal{A}_{B_{11,12}} \end{bmatrix} \begin{pmatrix} w_{11} \\ w_{12} \\ w_{B_{11,12}} \end{pmatrix} = \begin{pmatrix} \phi_{11} \\ \phi_{12} \\ \phi_{B_{11,12}} \end{pmatrix}$$

and

$$\begin{bmatrix} \mathcal{A}_{21} & 0 & \mathcal{A}_{1B_{21,22}} \\ 0 & \mathcal{A}_{22} & \mathcal{A}_{2B_{21,22}} \\ \mathcal{A}_{1B_{21,22}}^T & \mathcal{A}_{2B_{21,22}}^T & \mathcal{A}_{B_{21,22}} \end{bmatrix} \begin{pmatrix} w_{21} \\ w_{22} \\ w_{B_{21,22}} \end{pmatrix} = \begin{pmatrix} \phi_{21} \\ \phi_{22} \\ \phi_{B_{21,22}} \end{pmatrix},$$

where Ω_1 is subdivided in Ω_{11} and Ω_{12} and Ω_2 is subdivided in Ω_{21} and Ω_{22}. This decomposition can be applied until the desired number of subdomains is reached, see Figure 4.5.



(a) 2 subdomains (b) 4 subdomains

Figure 4.5: Recursive domain decomposition

This procedure is showcase formulated for the case of four nonoverlapping subdomains in Algorithm 4.4. An application for even more subdomains is straightforward.

Algorithm 4.4 Recursive multi-subdomain algorithm, 4 subdomains

Let Ω be decomposed into $\Omega_1 \cup B_{1,2} \cup \Omega_2$, where Ω_1 and Ω_2 are further decomposed into $\Omega_1 = \Omega_{11} \cup B_{11,12} \cup \Omega_{12}$ and $\Omega_2 = \Omega_{21} \cup B_{21,22} \cup \Omega_{22}$. Let $n \times m_{ij}$, $i,j = 1,2$ denote the number of internal grid points of Ω_{ij}.

1: Compute $\tilde{\phi}_{B_{1,2}} = \phi_{B_{1,2}} - \mathcal{A}^T_{1B_{1,2}} \mathcal{A}_1^{-1} \phi_1 - \mathcal{A}^T_{2B_{1,2}} \mathcal{A}_2^{-1} \phi_2$
where $w_1 = \mathcal{A}_1^{-1}\phi_1$ and $w_2 = \mathcal{A}_2^{-1}\phi_2$ are computed recursively by domain decomposition:

a) Compute
$-\mathcal{A}^T_{11B_{11,12}} \mathcal{A}_{11}^{-1} \phi_{11}, \quad -\mathcal{A}^T_{12B_{11,12}} \mathcal{A}_{12}^{-1} \phi_{12}, \quad -\mathcal{A}^T_{21B_{21,22}} \mathcal{A}_{21}^{-1} \phi_{21}, \quad -\mathcal{A}^T_{22B_{21,22}} \mathcal{A}_{22}^{-1} \phi_{22}$
and set

$$\tilde{\phi}_{B_{11,12}} = \phi_{B_{11,12}} - \mathcal{A}^T_{11B_{11,12}} \mathcal{A}_{11}^{-1} \phi_{11} - \mathcal{A}^T_{12B_{11,12}} \mathcal{A}_{12}^{-1} \phi_{12}$$
$$\tilde{\phi}_{B_{21,22}} = \phi_{B_{21,22}} - \mathcal{A}^T_{21B_{21,22}} \mathcal{A}_{21}^{-1} \phi_{21} - \mathcal{A}^T_{22B_{21,22}} \mathcal{A}_{22}^{-1} \phi_{22}.$$

b) Solve $S w_{B_{11,12}} = \tilde{\phi}_{B_{11,12}}$ with $S = S(n, m_{11}, m_{12})$
and $S w_{B_{21,22}} = \tilde{\phi}_{B_{21,22}}$ with $S = S(n, m_{21}, m_{22})$.

c) Compute

$$w_{11} = \mathcal{A}_{11}^{-1}(\phi_{11} - \mathcal{A}_{1B_{11,12}} w_{B_{11,12}}),\ w_{12} = \mathcal{A}_{12}^{-1}(\phi_{12} - \mathcal{A}_{2B_{11,12}} w_{B_{11,12}})$$
$$\text{and } w_{21} = \mathcal{A}_{21}^{-1}(\phi_{21} - \mathcal{A}_{1B_{21,22}} w_{B_{21,22}}),\ w_{22} = \mathcal{A}_{22}^{-1}(\phi_{22} - \mathcal{A}_{2B_{21,22}} w_{B_{21,22}}).$$

d) Set $w_1 = [w_{11}, w_{B_{11,12}}, w_{12}]^T$, $w_2 = [w_{21}, w_{B_{21,22}}, w_{22}]^T$
and $\tilde{\phi}_{B_{1,2}} = \phi_{B_{1,2}} - \mathcal{A}^T_{1,B_{1,2}} w_1 - \mathcal{A}^T_{2,B_{1,2}} w_2$.

2: Solve $S w_{B_{1,2}} = \tilde{\phi}_{B_{1,2}}$ with $S = S(n, m_{11} + m_{12} + 1, m_{21} + m_{22} + 1)$.

3: Compute $w_1 = \mathcal{A}_1^{-1}(\phi_1 - \mathcal{A}_{1B_{1,2}} w_{B_{1,2}})$ and $w_2 = \mathcal{A}_2^{-1}(\phi_2 - \mathcal{A}_{2B_{1,2}} w_{B_{1,2}})$ recursively.
Set $\phi_1 = \phi_1 - \mathcal{A}_{1B_{1,2}} w_{B_{1,2}}$ and $\phi_2 = \phi_2 - \mathcal{A}_{2B_{1,2}} w_{B_{1,2}}$ and proceed as in Step 1:

a) Compute
$-\mathcal{A}^T_{11B_{11,12}} \mathcal{A}_{11}^{-1} \phi_{11}, \quad -\mathcal{A}^T_{12B_{11,12}} \mathcal{A}_{12}^{-1} \phi_{12}, \quad -\mathcal{A}^T_{21B_{21,22}} \mathcal{A}_{21}^{-1} \phi_{21}, \quad -\mathcal{A}^T_{22B_{21,22}} \mathcal{A}_{22}^{-1} \phi_{22}$
and set

$$\tilde{\phi}_{B_{11,12}} = \phi_{B_{11,12}} - \mathcal{A}^T_{11B_{11,12}} \mathcal{A}_{11}^{-1} \phi_{11} - \mathcal{A}^T_{12B_{11,12}} \mathcal{A}_{12}^{-1} \phi_{12}$$
$$\tilde{\phi}_{B_{21,22}} = \phi_{B_{21,22}} - \mathcal{A}^T_{21B_{21,22}} \mathcal{A}_{21}^{-1} \phi_{21} - \mathcal{A}^T_{22B_{21,22}} \mathcal{A}_{22}^{-1} \phi_{22}.$$

b) Solve $S w_{B_{11,12}} = \tilde{\phi}_{B_{11,12}}$ with $S = S(n, m_{11}, m_{12})$
and $S w_{B_{21,22}} = \tilde{\phi}_{B_{21,22}}$ with $S = S(n, m_{21}, m_{22})$.

c) Compute

$$w_{11} = \mathcal{A}_{11}^{-1}(\phi_{11} - \mathcal{A}_{1B_{11,12}} w_{B_{11,12}}),\ w_{12} = \mathcal{A}_{12}^{-1}(\phi_{12} - \mathcal{A}_{2B_{11,12}} w_{B_{11,12}})$$
$$\text{and } w_{21} = \mathcal{A}_{21}^{-1}(\phi_{21} - \mathcal{A}_{1B_{21,22}} w_{B_{21,22}}),\ w_{22} = \mathcal{A}_{22}^{-1}(\phi_{22} - \mathcal{A}_{2B_{21,22}} w_{B_{21,22}}).$$

d) Set $w_1 = [w_{11}, w_{B_{11,12}}, w_{12}]^T$, $w_2 = [w_{21}, w_{B_{21,22}}, w_{22}]^T$
and $\tilde{\phi}_{1,2} = \phi_{B_{1,2}} - \mathcal{A}^T_{1,B_{1,2}} w_1 - \mathcal{A}^T_{2,B_{1,2}} w_2$.

4: Set $w = [w_1, w_{B_{1,2}}, w_2]^T$.

The four subdomain solution procedures in Step 1a can be done in parallel as well as the two solutions of the Schur complements in Step 1b and the four solutions in Step 1c. The same holds for the computations in Step 3a-3c. A more detailed performance consideration will be given in Section 4.5.3.

4.5.2 Simultaneous Solver

The second possibility considered is solving the complete Schur Complement system with an iterative or direct method. Let the size of the grid on Ω be $n \times m$. After decomposition into strips, let there be $n \times m_i$ grids on each subdomain Ω_i. The Schur complement

$$S = \mathcal{A}_B - \mathcal{A}_{IB}^T \mathcal{A}_I^{-1} \mathcal{A}_{IB}$$

of

$$\begin{bmatrix} \mathcal{A}_I & \mathcal{A}_{IB} \\ \mathcal{A}_{IB}^T & \mathcal{A}_B \end{bmatrix} \begin{pmatrix} w_I \\ w_B \end{pmatrix} = \begin{pmatrix} \phi_I \\ \phi_B \end{pmatrix}$$

has the specific form

$$S = \begin{bmatrix} H_1 & Q_2 & & \\ Q_2 & H_2 & \ddots & \\ & \ddots & \ddots & Q_{k-1} \\ & & Q_{k-1} & H_{k-1} \end{bmatrix}$$

where

$$H_i = \mathcal{A}_{B_{i,i+1}} - \mathcal{A}_{iB_{i,i+1}}^T \mathcal{A}_i^{-1} \mathcal{A}_{iB_{i,i+1}} - \mathcal{A}_{i+1B_{i,i+1}}^T \mathcal{A}_{i+1}^{-1} \mathcal{A}_{i+1B_{i,i+1}}$$

and

$$Q_i = -\mathcal{A}_{i-1B_{i-1,i}}^T \mathcal{A}_i^{-1} \mathcal{A}_{iB_{i,i+1}}.$$

For each of the blocks H_i and Q_i an exact decomposition is available similar to the two subdomains case. H_i is the Schur complement matrix for the two subdomain case of the subdomains Ω_i, Ω_{i+1} with the interface $B_{i,i+1}$ and Q_i takes account of the coupling of the interfaces $B_{i-1,1}$, and $B_{i,i+1}$. The exact decomposition of Q_i can be computed as in the preceding section. The formal description of the algorithm is given below.

Algorithm 4.5 Simultaneous multi-domain algorithm

Let Ω be decomposed into k strips as described above.

1: Compute $\tilde{\phi}_B = \phi_B - \mathcal{A}_{IB}^T \mathcal{A}_I^{-1} \phi_I$.
2: Set up the Schur complement S and solve $S w_B = \tilde{\phi}_B$.
3: Compute $w_I = \mathcal{A}_I^{-1}(\phi_I - \mathcal{A}_{IB} w_B)$.

For a comparison with the recursive approach, the simultaneous algorithm is formulated in Algorithm 4.6 for the case of four subdomains and with the decomposition in strips, illustrated in Figure 4.6.

The main work is the solution of the linear system in Step 2 of Algorithm 4.6. For every block $H_i, i = 1, \ldots, k-1$, and $Q_j, j = 2, \ldots, k-1$, an exact decomposition is available and the corresponding matrices are easily to invert. Each block is tri-blockdiagonal and positive definite, but not symmetric. A fast solver is needed for this simultaneous approach. In this thesis this linear system is solved with a BLOCKLU decomposition [35, Chapter 2.3.3].

Algorithm 4.6 Simultaneous multi-domain algorithm, 4 subdomains

Let Ω be decomposed into four strips Ω_i each with $n \times m_i$, $i = 1, \ldots, 4$ internal grid points and the interfaces $B_{1,2}$, $B_{2,3}$, $B_{3,4}$.

1: Compute

$$-\mathcal{A}_{1B_{1,2}}^T \mathcal{A}_1^{-1}\phi_1, -\mathcal{A}_{2B_{1,2}}^T \mathcal{A}_2^{-1}\phi_2, -\mathcal{A}_{2B_{2,3}}^T \mathcal{A}_2^{-1}\phi_2,$$
$$-\mathcal{A}_{3B_{2,3}}^T \mathcal{A}_3^{-1}\phi_3, -\mathcal{A}_{3B_{3,4}}^T \mathcal{A}_3^{-1}\phi_3 - \mathcal{A}_{4B_{3,4}}^T \mathcal{A}_4^{-1}\phi_4$$

and set

$$\tilde{\phi}_{B_{1,2}} = \phi_{B_{1,2}} - \mathcal{A}_{1B_{1,2}}^T \mathcal{A}_1^{-1}\phi_1 - \mathcal{A}_{2B_{1,2}}^T \mathcal{A}_2^{-1}\phi_2,$$
$$\tilde{\phi}_{B_{2,3}} = \phi_{B_{2,3}} - \mathcal{A}_{2B_{2,3}}^T \mathcal{A}_2^{-1}\phi_2 - \mathcal{A}_{3B_{2,3}}^T \mathcal{A}_3^{-1}\phi_3,$$
$$\tilde{\phi}_{B_{3,4}} = \phi_{B_{3,4}} - \mathcal{A}_{3B_{3,4}}^T \mathcal{A}_3^{-1}\phi_3 - \mathcal{A}_{4B_{3,4}}^T \mathcal{A}_4^{-1}\phi_4.$$

2: Compute

$$H_1 = \mathcal{A}_{B_{1,2}} - \mathcal{A}_{1B_{1,2}}^T \mathcal{A}_1^{-1}\mathcal{A}_{1B_{1,2}} - \mathcal{A}_{2B_{1,2}}^T \mathcal{A}_2^{-1}\mathcal{A}_{2B_{1,2}},$$
$$H_2 = \mathcal{A}_{B_{2,3}} - \mathcal{A}_{2B_{2,3}}^T \mathcal{A}_2^{-1}\mathcal{A}_{2B_{2,3}} - \mathcal{A}_{3B_{2,3}}^T \mathcal{A}_3^{-1}\mathcal{A}_{3B_{2,3}},$$
$$H_3 = \mathcal{A}_{B_{3,4}} - \mathcal{A}_{3B_{3,4}}^T \mathcal{A}_3^{-1}\mathcal{A}_{3B_{3,4}} - \mathcal{A}_{4B_{3,4}}^T \mathcal{A}_4^{-1}\mathcal{A}_{4B_{3,4}},$$

and

$$Q_2 = -\mathcal{A}_{1B_{1,2}}^T \mathcal{A}_2^{-1}\mathcal{A}_{2B_{2,3}},$$
$$Q_3 = -\mathcal{A}_{2B_{2,3}}^T \mathcal{A}_3^{-1}\mathcal{A}_{3B_{3,4}}.$$

Set up the Schur complement

$$S = \begin{bmatrix} H_1 & Q_2 & 0 \\ Q_2 & H_2 & Q_3 \\ 0 & Q_3 & H_3 \end{bmatrix}.$$

Solve

$$\begin{bmatrix} H_1 & Q_2 & 0 \\ Q_2 & H_2 & Q_3 \\ 0 & Q_3 & H_3 \end{bmatrix} \begin{pmatrix} w_{B_{1,2}} \\ w_{B_{2,3}} \\ w_{B_{3,4}} \end{pmatrix} = \begin{pmatrix} \tilde{\phi}_{B_{1,2}} \\ \tilde{\phi}_{B_{2,3}} \\ \tilde{\phi}_{B_{3,4}} \end{pmatrix}.$$

3: Compute

$$w_1 = \mathcal{A}_1^{-1}(\phi_1 - \mathcal{A}_{1,B_{1,2}} w_{B_{1,2}}),$$
$$w_2 = \mathcal{A}_2^{-1}(\phi_2 - \mathcal{A}_{1,B_{1,2}} w_{B_{1,2}} - \mathcal{A}_{2,B_{2,3}} w_{B_{2,3}}),$$
$$w_3 = \mathcal{A}_3^{-1}(\phi_3 - \mathcal{A}_{2,B_{2,3}} w_{B_{2,3}} - \mathcal{A}_{3,B_{3,4}} w_{B_{3,4}}),$$
$$w_4 = \mathcal{A}_4^{-1}(\phi_4 - \mathcal{A}_{3,B_{3,4}} w_{B_{3,4}}).$$

4: Set $w = [w_1, w_{B_{1,2}}, w_2, w_{B_{2,3}}, w_3, w_{B_{3,4}}, w_4]^T$.

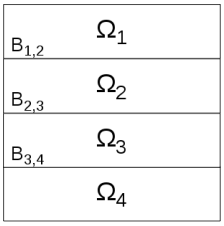


Figure 4.6: Simultaneous domain decomposition

4.5.3 Performance Consideration

In this section, a survey of the amount of computational work, the demand of memory as well as the degree of parallelization is given. In the first place one has to distinguish between *offline* and *online costs*. Offline costs is the work that can be computed in the preprocessing stage as these results can be reused several times for various problems and e.g. do not depend on the right-hand side. In the case of the recursive approach, the setup of the Schur complement matriced are considered as offline costs and for the simultaneous approach, the setup of the BLOCKLU decomposition is done in the preprocessing. The remaining work, like subdomain solves and the matrix-vector products for with the capacitance matrix for determining the unknowns on the interfaces, are online costs. Firstly the offline costs of both approaches are considered for the constant coefficent case with a uniform decomposition, i.e. all subdomains have the same size in x_2-direction and a non-uniform decomposition. In the uniform case, a lot of efforts can be saved as the Schur complement depends only on the sizes of the subdomains, and once assembled, it can be reused for similar decompositions. For every Schur complement setup three steps are necessary:

1. Compute the diagonals of Λ.
2. Determine the inverse of Λ.
3. Compute $\bar{F}\Lambda^{-1}\bar{F}$.

For the example of the four subdomain algorithm, the following considerations for the computational complexity and the demand of memory hold. Let the domain Ω be decomposed into four subdomains with a $n \times m_{ij}$ grid with $\sum_{i,j=1}^{2} m_{ij} = m$ resp. on an $n \times m_i$ grid with $\sum_{i=1}^{4} m_i = m$.

	recursive		simultaneous	
	non-uniform	uniform	non-uniform	uniform
# Schur complements	3	1	7	2
Memory usage	$3(2n \times 2n)$	$2n \times 2n$	$7(2n \times 2n)$	$2(2n \times 2n)$

Table 4.1: Offline costs recursive vs. simultaneous, 4 Subdomains

The Table 4.2 compares the online costs of the two approaches. Here only the subdomain solves (SD) and the matrix-vector products (MV) to determine the interface unknowns are considered.

	recursive		simultaneous	
step	non-uniform	uniform	non-uniform	uniform
1a	4 SD, $n \times m_{ij}$	4 SD, $n \times \frac{m}{4}$	4 SD, $n \times m_i$	4 SD, $n \times m_i$
1b	2 MV	6 MV		
1c	4 SD, $n \times m_{ij}$	4 SD, $n \times \frac{m}{4}$		
2	1 MV	1 MV	5 SC	2 SC
			solve linear system (LS)	solve linear system (LS)
3a	4 SD, $n \times m_{ij}$	4 SD, $n \times \frac{m}{4}$	4 SD, $n \times m_i$	4 SD, $n \times m_i$
3b				
3c	4 SD, $n \times m_{ij}$	4 SD, $n \times \frac{m}{4}$		
$\sum$	16 SD, 3 SC	16 SD, 2 SC	8 SD, 5 SC + LS	8 SD, 2 SC + LS

Table 4.2: Online costs recursive vs. simultaneous, 4 Subdomains

Table 4.3 compares the amount of work for increasing numbers of subdomains.

	recursive				simultaneous			
	non-uniform		uniform		non-uniform		uniform	
#	SD	SC	SD	SC	SD	SC	SD	SC
1	1	0	1	0	1	0	1	0
2	4	1	4	1	4	1	4	1
4	16	3	10	2	8	5	8	2
8	64	7	64	3	16	13	16	2
k	k^2	$k-1$	k^2	$\log_2(k)$	$2k$	$2k-3$	$2k$	2

Table 4.3: Amount of work for increasing numbers of subdomains

Disregarding the Schur complement effort and considering the uniform decomposition, for the recursive approach it holds that

- for k subdomains, k^2 solutions on a grid sized $n \times \frac{m}{k}$ have to be performed with a amount of work roughly calculated by k solves on $n \times m$

and for the simultaneous approach it holds that

- for k subdomains, $2k$ solutions on a grid sized $n \times \frac{m}{k}$ have to be performed with a amount of work roughly calculated by 2 solves on $n \times m$.

Because of the design of the algorithm, these k solves can be done in parallel. Assumed that the Schur complement setup part is neglected, it turns out that a performance analysis for the simultaneous approach depends on the efficiency of the solution of the linear system.
Another point to consider is the increase of the number of processors and the use of the parallel possibilities. The following performance considerations hold for the online costs:
For the recursive domain decomposition the theoretical calculation

$$\frac{\text{effort}(n_{proc} \cdot V_0)}{n_{proc}} = n_{proc} \cdot const,$$

and for the simultaneous approach

$$\frac{\text{effort}(n_{proc} \cdot V_0)}{n_{proc}} = const.$$

is valid, where V_0 denotes the size of the entire domain and n_{proc} the number of processors. As a result, the elapsed time should stay constant if using k processors for k subdomains when setting up the Schur complement matrices in the preprocessing stage. This is a calculation for an ideal algorithm, which scales perfectly and neglecting costs for communication, data transfer and technical characteristics.

4.6 Extensions to more General Problems

In this section possible extensions of the exact decomposition of the domain decomposition interface operators for constant coefficient systems of elliptic operators are considered. Firstly, a straightforward three-dimensional version is presented in Section 4.6.1. Another possible use of the Schur complement is as a direct preconditioner for variable coefficient problems on more general domains, considered in Chapter 4.6.2 and Chapter 4.6.3.

4.6.1 Three-dimensional Problems

Let Ω_h be a uniform mesh grid on a regular three-dimensional domain Ω with grid size h in x_1-direction, i.e.

$$h = \frac{1}{n_1 + 1},$$

where n_1 is the number of grid points in x_1-direction. Let n_2 be the number of grid points in x_2-direction and m the number of grid points in x_3-direction. The domain is divided into two three-dimensional 'slices', with m_1 resp. m_2 internal grid points in Ω_1 and Ω_2, where m_1 and m_2 are assumed as integral multiples of h and the interface B parallel to the (x_1, x_2)-axis, see Figure 4.7.

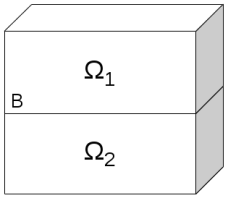


Figure 4.7: Domain decomposition in three dimensions

Let the uniformly elliptic operator L with constant coefficients be approximated by the finite difference discretization in a general form:

$$\begin{aligned} Lu = a(x_1 - h, x_2, x_3) + bu(x_1, x_2, x_3) + cu(x_1 + h, x_2, x_3) + du(x_1, x_2 - h, x_3) \\ + eu(x_1, x_2 + h, x_3) + fu(x_1, x_2, x_3 - h) + gu(x_1, x_2, x_3 + h). \end{aligned}$$

In stencil notation one has

$$\begin{bmatrix} & f & \end{bmatrix}_h \begin{bmatrix} & d & \\ a & b & c \\ & e & \end{bmatrix}_h \begin{bmatrix} & g & \end{bmatrix}_h,$$

where the coefficients are assumed to be nonzero. Analogously to the two-dimensional case, an exact decomposition can be derived.

THEOREM 4.9 (Exact decomposition of S in three dimensions)**.**
The Schur complement matrix S can be decomposed into

$$S = \bar{F}\Lambda\bar{F}^{-1}$$

where

$$\bar{F} = \begin{pmatrix} \tilde{F} & 0 \\ 0 & \tilde{F} \end{pmatrix}, \tilde{F} = D_{n_2}F_{n_2} \otimes D_{n_1}F_{n_1}$$

with

$$(F_{n_k})_{ij} = \sqrt{2h}\sin(ij\pi h), i, j = 1, \ldots, n_k.$$

D_1, D_2 are diagonal matrices, where

$$(D_{n_1})_{ii} = \sqrt{\frac{a}{c}}^i, i = 1, \ldots, n_1$$

and

$$(D_{n_2})_{ii} = \sqrt{\frac{d}{e}}^i, i = 1, \ldots, n_2.$$

$X \otimes Y$ is defined as a *direct* (or *tensor*) *product* with the following two properties

1. $(X \otimes Y)^T = X^T \otimes Y^T$
2. $(X_1 \otimes Y_1)(X_2 \otimes Y_2) = (X_1X_2) \otimes (Y_1Y_2)$.

The matrix $\bar{F}$ can easily be inverted:

$$\begin{aligned}(D_{n_2}F_{n_2} \otimes D_{n_1}F_{n_1})^{-1} &= ((D_{n_2} \otimes D_{n_1})(F_{n_2} \otimes F_{n_1}))^{-1} \\ &= (F_{n_2} \otimes F_{n_1})^{-1}(D_{n_2} \otimes D_{n_1})^{-1} \\ &= (F_{n_2}^{-1} \otimes F_{n_1}^{-1})(D_{n_2}^{-1} \otimes D_{n_1}^{-1}) \\ &= (F_{n_2}D_{n_2}^{-1}) \otimes (F_{n_1}D_{n_1}^{-1}).\end{aligned}$$

The derivation of the tri-blockdiagonal matrix Λ is analogous to the two-dimensional case. Simple calculation shows that

$$\begin{aligned}\mathcal{A}_B &= \bar{F}\Lambda_B\bar{F}^{-1} \\ &= \begin{bmatrix} \alpha\,\mathrm{diag}(b + \sqrt{ac}(2 - \sigma_i) + \sqrt{de}(2 - \sigma_j)) & I_{n_1n_2} \\ -I_{n_1n_2} & \mathrm{diag}(b + \sqrt{ac}(2 - \sigma_i) + \sqrt{de}(2 - \sigma_j)) \end{bmatrix}\end{aligned}$$

for $i = 1, \ldots, n_1$ and $j = 1, \ldots, n_2$.

REMARK 4.10.
A diagonal matrix $\mathrm{diag}(a_{ij})$, $i = 1, \ldots, n_1$ and $j = 1, \ldots, n_2$, depending on two indices is enumerated as

$$\mathrm{diag}(a_{ij}) := \mathrm{diag}(b_k), \quad \text{where } b_k = a_{(i-1)+(j-1)n_1+1}, k = 1, \ldots, n_1n_2.$$

The contribution of the term $-\mathcal{A}_{1B}^T\mathcal{A}_1^{-1}\mathcal{A}_{1B}$ can be determined by solving the systems

$$\alpha[(b+\sqrt{ac}(2-\sigma_i)+\sqrt{de}(2-\sigma_j))\bar{d}_k+f\bar{d}_{k-1}+g\bar{d}_{k+1}]-\bar{e}_k=0$$
$$[(b+\sqrt{ac}(2-\sigma_i)+\sqrt{de}(2-\sigma_j))\bar{e}_k+f\bar{e}_{k-1}+g\bar{e}_{k+1}]+\bar{d}_k=0$$

with boundary conditions

$$\bar{d}_0=1,\bar{e}_0=0,\bar{d}_{m_1+1}=0,\bar{e}_{m_1+1}=0$$

which have to be solved for $i=1,\dots,n_1$ and $j=1,\dots,n_2$.
Similarly for $-\mathcal{A}_{2B}^T\mathcal{A}_2^{-1}\mathcal{A}_{2B}$ one has to solve

$$\alpha[(b+\sqrt{ac}(2-\sigma_i)+\sqrt{de}(2-\sigma_j))\mathring{d}_k+f\mathring{d}_{k-1}+g\mathring{d}_{k+1}]-\mathring{e}_k=0$$
$$[(b+\sqrt{ac}(2-\sigma_i)+\sqrt{de}(2-\sigma_j))\mathring{e}_k+f\mathring{e}_{k-1}+g\mathring{e}_{k+1}]+\mathring{d}_k=0$$

with boundary conditions

$$\mathring{d}_0=1,\mathring{e}_0=0,\mathring{d}_{m_1+1}=0,\mathring{e}_{m_1+1}=0$$

for $i=1,\dots,n_1$ and $j=1,\dots,n_2$.
Summarized, this leads to

$$\Lambda=\begin{bmatrix}\alpha\,\mathrm{diag}(\eta_{ij}) & \mathrm{diag}(1-f\bar{e}_1^{ij}+g\mathring{e}_1^{ij})\\ -\mathrm{diag}(1-f\bar{e}_1^{ij}+g\mathring{e}_1^{ij}) & \mathrm{diag}(\eta_{ij})\end{bmatrix}$$

where $\eta_{ij}=b+\sqrt{ac}(2-\sigma_i)+\sqrt{de}(2-\sigma_j)+f\bar{d}_1^{ij}+g\mathring{d}_1^{ij}$.

EXAMPLE 4.11 (Laplace 3D)**.**
Consider the optimality system $\mathcal{A}$ of the academic optimal control problem with the Laplace operator $L=-\Delta$ and $\Omega=(0,1)^3$ divided into $\Omega_1=(0,1)\times(0,1)\times(0,0.5)$, $\Omega_2=(0,1)\times(0,1)\times(0.5,1)$ and $B=(0,1)\times(0,1)\times(0.5)$. The difference stencil is given by

$$-\frac{1}{h^2}\begin{bmatrix}&&\\&1&\\&&\end{bmatrix}_h\begin{bmatrix}&1&\\1&-6&1\\&1&\end{bmatrix}_h\begin{bmatrix}&&\\&1&\\&&\end{bmatrix}_h.$$

As $\Omega_1=\Omega_2$ and as the Schur complement only depends on the size of the subdomains $\bar{d}_1^{ij}=\mathring{d}_1^{ij}(:=d_1^{ij})$ and $\bar{e}_1^{ij}=\mathring{e}_1^{ij}(:=e_1^{ij})$, the Schur complement has the form

$$S=F\Lambda F$$

where

$$\Lambda=\begin{bmatrix}-\frac{\alpha}{h^2}\,\mathrm{diag}(2d_1^{ij}-(2+\sigma_i+\sigma_j)) & -\frac{1}{h^2}\,\mathrm{diag}(2e_1^{ij}-h^2)\\ \frac{1}{h^2}\,\mathrm{diag}(2e_1^{ij}-h^2) & -\frac{1}{h^2}\,\mathrm{diag}(2d_1^{ij}-(2+\sigma_i+\sigma_j))\end{bmatrix}.$$

REMARK 4.12.

1. This procedure can be extended straightforward to the case of many subdomains as in the previous section by using the recursive or the simultaneous approach.
2. For every grid point on the 2D-interface, the solution of two problems is required and the whole Schur complement matrix has to be stored with a size of $2n_1n_2\times 2n_1n_2$. For the case of many subdomains, where several capacitance matrices for the purpose of reuse are set up, this is hard to handle because of limited memory.

4.6.2 More General Problems

For the derivation of the capacitance matrix the case of a uniformly elliptic operator L with constant coefficient on the whole domain was considered. This proposition still holds for operators with jumps on horizontal or vertical interfaces and the coefficients being constant on each subdomain Ω_i. The five-point discretization formula can be applied to these discontinuities as long as the coefficients on the interface are defined by arithmetic averages. Let Lu be given by

$$Lu = -a(x_1, x_2)\frac{\partial^2}{\partial x_1^2}u(x_1, x_2) - b(x_1, x_2)\frac{\partial^2}{\partial x_2^2}u(x_1, x_2) + c(x_1, x_2)u(x_1, x_2).$$

Continuity conditions require the continuity of the solution $u(x_1, x_2)$ in Ω, $b(x_1, x_2)u_y(x_1, x_2)$ to be continuous along horizontal interfaces and $a(x_1, x_2)u_x(x_1, x_2)$ to be continuous along vertical interfaces. For a point (x_1, x_2) at a horizontal interface redefine

$$a(x_1, x_2) = \frac{1}{2}(\lim_{x_2 \to x_2^-} a(x_1, x_2) + \lim_{x_2 \to x_2^+} a(x_1, x_2))$$

and similarly for a point (x_1, x_2) on vertical interfaces

$$b(x_1, x_2) = \frac{1}{2}(\lim_{x_1 \to x_1^-} b(x_1, x_2) + \lim_{x_1 \to x_1^+} b(x_1, x_2))$$

and for (x_1, x_2) being a cross point

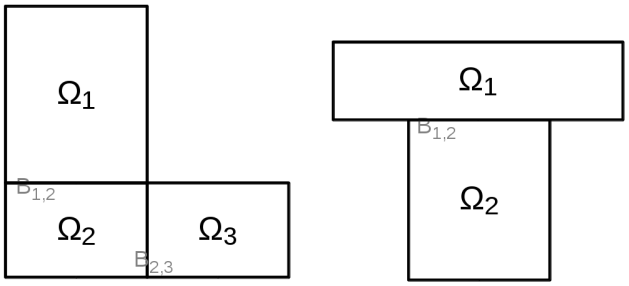
$$c(x_1, x_2) = \frac{1}{4}(\lim_{\substack{x_1 \to x_1^- \\ x_2 \to x_2^-}} c(x_1, x_2) + \lim_{\substack{x_1 \to x_1^- \\ x_2 \to x_2^+}} c(x_1, x_2) + \lim_{\substack{x_1 \to x_1^+ \\ x_2 \to x_2^-}} c(x_1, x_2) + \lim_{\substack{x_1 \to x_1^+ \\ x_2 \to x_2^+}} c(x_1, x_2)),$$

see [47, Chapter 2.3] or [55, p. 169ff].
More general differential operators with variable coefficients can be approximated by constant operators, for example by averaging the coefficients over the subdomains. In this case, the Schur complement matrix of a constant differential operator can be used as a direct preconditioner for variable coefficient problems, see [47, Chapter 7], [21], where the Schur complement method is applied for the solution of the Laplace problem. Preconditioning is useful, as the rate of convergence depends on the variations in the coefficients and in general, when solving the capacitance system by an iterative method, every iteration needs the solution of a problem on each subdomain. For efficiency reasons, it is essential to keep the number of iterations low by using good preconditioners. Another straightforward extension is the treatment of more general boundary conditions. For a derivation of the capacitance system within the finite element context, see [24].

4.6.3 Irregular Domains

For general irregular domains, the exact decomposition cannot be computed analytically using the techniques within the presented framework. Therefore, for solving the equation $Sw_B = \phi_B$, iterative methods like preconditioned Krylov-subspace methods have to be applied. The application of such methods requires only products of matrices and vectors. Nevertheless domain

decompositions can also be applied to more general (irregular) domains. In the first place, one can consider an irregular mesh, where the mesh width in the horizontal direction differs from the mesh width in vertical direction. Therefore an exact decomposition can still be derived, see [47]. Furthermore, if an irregular domain can be decomposed in regular domains,



(a) L-shaped (b) T-shaped

Figure 4.8: Irregular domains

like L- or T-shaped domains, see Figure 4.8, which are decomposed in rectangulars, a preconditioner can be derived. An L-shaped domain is the simplest one that can be decomposed into rectangular subdomains. There are two possibilities to divide the domain into two rectangles: $\Omega_1 \cup \Omega_2$ and Ω_3 with the interface $B_{2,3}$ or Ω_1 and $\Omega_2 \cup \Omega_3$ with the interface $B_{1,2}$. The idea is to take the two-subdomain Schur complement $M_{B_{1,2}}$ resp. $M_{B_{2,3}}$, given by the exact decomposition for the case that the subdomain Ω_3 resp. Ω_1 was absent, as preconditioner for the resulting systems

$$S_{B_{1,2}} w_{B_{1,2}} = \phi_{B_{1,2}} \quad \text{and} \quad S_{B_{2,3}} w_{B_{2,3}} = \phi_{B_{2,3}},$$

in particular

$$\tilde{S}_{B_{1,2}} = M_{B_{1,2}}^{-1} S_{B_{1,2}} \quad \text{and} \quad \tilde{S}_{B_{2,3}} = M_{B_{2,3}}^{-1} S_{B_{2,3}}.$$

Chan and Resasco showed in [47], [21] that in the general case there is a priori no reason why to prefer one way of decomposition to the other one as both systems are equivalent from the convergence point of view for the case that a domain decomposition for the Poisson equation is done. The rate of convergence depends on the *aspect ratios* of the subdomains. This result should stay valid for the case of the optimality system.
For more irregular domains, the idea is to approximate the irregular domains by domains with regular geometries that share the same interface. Chan [17] suggested a procedure for extending the exact preconditioner for rectangular regions to construct preconditioners for the Schur complement of irregular domains. The idea is to define a preconditioner for the interface that uses the exact capacitance matrix corresponding to the best rectangular approximations to the irregular domain sharing the same interface.

5 GPU Programming

Due to the well-known Moore's law the number of transistors that can be placed on an integrated circuit doubles approximately every 18-24 month. This rule of thumb was an observation by Gordon E. Moore, one of the Intel Corporation co-founders, and proposed in 1965 [41]. This prediction has become a benchmark in the research and development of semiconductors. Doubling the number of transistors leads, beyond growth in other hardware aspects like more on-chip memory, more memory management units or more specialized hardware, to higher clock frequencies. Increasing the speed of operation of a processor clock is operating has been the most important method for the last 30 years. It turned out to be a reliable source for improving the performance of a single central processing unit (CPU). For example, a CPU ran at the beginning of the 1980's with a clock frequency of 1 MHz, growing up to 4 GHz on a recent INTEL XEON processor in 2012. But because of miscellaneous fundamental limitations in the production of these integrated circuits, like power and heat restrictions as well as the approaching physical limit to transistor size, a further increase of the clock rate and the amount of work that can be performed in each cycle clock period within a single CPU is limited. To achieve another performance improvement, the recent idea is to place more than one single processing core on a chip. Another approach is to use co-processors like FPGAs or graphics cards to enhance the functions of the CPU by offloading processor-intensive tasks from the main processors and therefore accelerating the system performance.
For designing microprocessors, two main trajectories have been followed according to [37]:

- the *multi-core trajectory*, which maintains the execution speed of sequential programs while moving into multiple cores, each of which is an out-of-order, multiple instruction issue processor implementing the full x86 instructions set, hyper-threading and designed to maximize the execution speed of sequential programs.
- the *many-core trajectory*, which focuses more on the execution throughput of parallel applications with a large number of much smaller cores. Every core is a heavily multithreaded, in-order, single-instruction issue processor that shares its control and its instruction cache with several other cores.

In this chapter, the concept of performing general purpose computations on a many-core *graphics processing unit (GPU)* is considered. A brief historical retrospection of the evolution of graphics cards to general purpose processing units just as a comparison of the different (parallel) programming paradigms of a CPU and a GPU is given in Section 5.1, followed by the architecture of a modern graphics card presented in Section 5.2, which deals with the specific NVIDIA CUDA architecture, the CUDA API and the programming language CUDA C. In Section 5.3, performance considerations are presented and the implementation of the CSMG is described. For further reading, a comprehensive introduction into GPU programming can be found in [37], [48] or the current CUDA Programming Guide 4.2 [43].

5.1 GPUs as Streamprocessors

GPUs can be considered as streamprocessors. In general, a streamprocessor is a co-processor, working on streams, where a stream is simply a set of data that requires similar computation and this processing can be done in a highly parallel way. The task of the GPU is to work on independent vertices and fragments, but many of them in parallel, which is effective, when many vertices or fragments are processed in the same way. In this sense, GPUs are streamprocessors.
In this section, a brief retrospective of the history of the GPU as well as a comparison of a GPU in contrast to a CPU is given. For more details see [37].

5.1.1 History of GPUs

From the *beginning 1980's to the early 1990's*, 2D display accelerators offered hardware-assisted bitmap operations to support in the display and usability of graphical operating system and accelerate the memory-intensive work of texture mapping and rendering polygons. The fixed-function pipelines were configurable, but not programmable. In professional computing, the enterprise Silicon Graphics used 3D graphics for scientific and technical visualization, like CAD, and in 1992 the OpenGL library as programming interface to its hardware was released. OpenGL is a standardized specification, defining a platform-independent application programming interface (API) for writing 3D applications.
In the *mid-1990s*, the request for 3D graphics for PC gaming raised. The main vendors NVIDIA, ATI and 3dfx launched graphics accelerators to satisfy this demand. As transform and lighting were already integral parts of the OpenGL graphics pipeline, increasingly more of the graphics pipeline moved away from the CPU and was implemented directly on the graphics processor. For example, the NVIDIA GeForce 256 pushed capabilities of consumer graphics hardware and performed transforming and lighting directly on the GPU. The functionality of the graphics pipeline can be found in [37, Chapter 2.1.1.].
In *2001*, the new DirectX 8.0 standard required that concordant hardware contains both programmable vertex and programmable pixel shading stages. With this DirectX API, the developers had for the first time some control of the exact computations that would be performed on their GPUs. As the shader pipeline of these modern graphics accelerators offers massive floating point computational power, researchers and programmers tried to use this for general purpose programming. The standard high-level shading languages, such as DirectX, OpenGL or Cg were the only way to interact with the GPU, so performing arbitrary computations were still subject to the constraints of programming within a graphics API. GPUs were designed to produce a color for every pixel on the screen using programmable arithmetic units called pixel shader. In general, a pixel shader uses its (x_1, x_2) position on the screen as well as some additional information, like input color, texture coordinates or other attributes, to combine various inputs in computing a final color. For general purpose computing, researchers tried to make their problems appear to the GPU to be traditional rendering. These efforts that used graphic APIs for general purpose computing were known as *GPGPU* programs. But there were several drawbacks:

- the programmer needs to have close knowledge of the graphic APIs and the GPU architecture

- problems had to be expressed in terms of vertex coordinates, textures and shader programs, which leads to increasing program complexity
- basic programming features such as random reads and writes to memory were not supported, which restricts the programming model
- double precision was not supported (until recently), so some scientific applications could not run on a GPU.

In *2006/2007*, NVIDIA developed the compute unified device architecture (CUDA). The previous generations partitioned their computing resources into separate vertex and pixel shaders, whereas CUDA installed a unified shader pipeline replacing these vertex and pixel pipelines, which allowed each arithmetic logic unit (ALU) on the chip being able to execute vertex, pixel and geometry and to perform general purpose computations. These unified shaders were utilized as a scalar thread processor becoming fully programmable with large instruction memory, instruction cache and instruction sequencing logic. The costs of these additional hardware added on the chip were reduced by having multiple shader processors to share their resources, like instruction cache and instruction sequency logic. As processing on a massive number of vertices or pixels in parallel, this design works well. The disadvantages mentioned above were remedied:

- A new general purpose parallel programming interface on the silicon chip serves the requests of CUDA programs and a programming language CUDA C, based on standard ANSI C with a relatively small number of keyword for labeling data-parallel functions and their associated data structures was introduced. Instead of programming dedicated graphics units with graphics APIs, the programmer can write C programs with CUDA extensions and target a general purpose, massively parallel processor without having to learn a new programming language. Programmers no longer need to use the graphics API to access the GPU parallel computing capabilities.
- Arbitrary read and write access to the memory, i.e. load and store instructions with random byte addressing capability to support the C requirements, as well as a software managed cache called shared memory and a barrier synchronization for inter-thread communication, were integrated. The need for programmers to manually manage vector registers was eliminated. The single-instruction multiple-thread (SIMT) execution model was introduced, where multiple independent threads execute concurrently using single instructions.
- The ALUs were conform to the IEEE requirements for single-precision floating point arithmetic and designed to use an instruction set adapted for general computation rather than specifically for graphics. The latest GPUs provide full IEEE double-precision support.

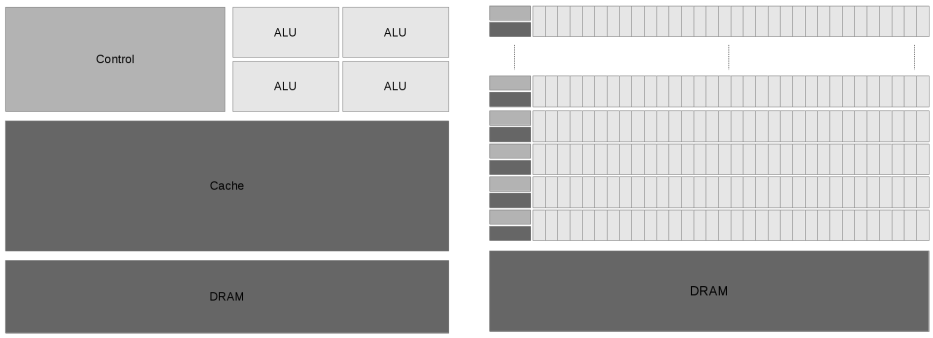
5.1.2 GPU vs. CPU

There is a huge gap between many-core GPUs and multi-core CPUs comparing the floating-point performance. In 2009, the ratio for this peak floating-point calculation throughput was about 10 to 1. [37, Figure 1.1]. This gap can be explained by the different *design philosophies*: The design of a *CPU* is optimized for sequential code performance. On a chip, about 90% of

the area is devoted to a sophisticated control logic that allows the reordering of instructions to extract best performance, out-of-order CPU execution to avoid delays waiting for read/write or earlier operations, branch predictions to minimize delays due to conditional branches and memory hierarchy with large cache to deliver data to register fast enough to feed the processor and reduce the instruction and data access latencies. The memory hierarchy consists of the slow, cheap and large main memory, L3, L1/L2 caches and registers, which are fast, but more expensive and small. The execution speed relies on exploiting data locality:

- *temporal locality:* data just accessed are kept in cache for later reuse
- *spatial locality:* neighboring data load in cache using a wide bus.

This is all handled automatically, the programmer does not need to mind about these caches, so it is simple but it is clear that in some cases, this does not extract the best performance. Multi-core CPUs follow the *MIMD (multiple instruction, multiple data)* parallel programming paradigm, that means that at any time, different processors may be executing different instructions on different pieces of data. Each core operates independently and each can be working with a different code, performing operations with completely different data. The INTEL CORE 2 processor has 4-8 MIMD cores, few registers, a big multi-level cache and a 10-30 GB/s bandwidth to the main memory.



(a) CPU (b) GPU

Figure 5.1: Different architectures according to [37]

The *GPU* has a different design philosophy. The chip area and power budget is maximized to perform floating-point calculations and optimized for execution throughput of massive numbers of threads minimizing control logic by having many threads available while waiting for long-latency memory accesses. A simplified logic compared to the CPU, for example there is no out-of-order execution and no branch prediction, allows that much more of the chip is devoted to floating-point computation. Small cache memories are provided to help to control the bandwidth requirements of these applications. Hence, multiple threads that can access the same memory data do not all need to go to the slow graphics memory (DRAM). The streamprocessors are arranged as multiple units, called *stream multi-processors (SM)* with each unit being effectively a vector unit. All cores of a SM are doing the same thing at the same time, so only one instruction decoder is needed to control all cores of a SM. Many-core GPUs follow the *SIMD (single instructions, multiple data)* parallel programming paradigm, that means all processors perform the same operation on multiple data simultaneously. All cores execute the same instruction at the same time, but they may be working on different data. The

most recent NVIDIA GTX 470 has 448 cores, arranged as 14 units each with 32 SIMD cores, lots of registers (16-48kB), almost no cache, 5GB/s bandwidth to host processor via the PCIe connection and 140GB/s bandwidth to the graphics memory. The different architectures are schematically described in Figure 5.1.
Both of GPU and CPU provide strengths and weaknesses for a particular scope of duties, so heterogeneous computing promises to achieve the best performance: sequential parts of the program that exhibit little or no data parallelism are implemented on the CPU, numerically intensive parts with a rich amount of data parallelism are implemented on the GPU.

5.2 GPU Computing

In contrast to the term GPGPU, general purpose computing in the CUDA architecture refers to the term *GPU computing*. The CUDA framework presented in this section is limited to NVIDIA GPUs. A more general model for writing programs that execute across heterogeneous platform consisting of CPUs and GPUs is the standard programming model OpenCL, defining language extensions for writing functions and a runtime API enabling programmers to manage parallelism and data delivery on massively parallel processors. OpenCL is a standardized programming model in which applications developed in OpenCL can run without any modification on all processors that support the OpenCL language extension, including heterogeneous CPU/GPU programming. Comparing CUDA with OpenCL on NVIDIA graphics card, a loss of performance of 5% to 50% is reported in [36].
The G80 chip was the first CUDA capable processor, integrated in the TESLA graphics cards. With it, NVIDIA introduced two key technologies: the unified graphics and compute architecture and CUDA, a software and hardware architecture that enabled the GPU to be programmed with a high level programming language. To non-graphics application programmers, the TESLA GPU introduced a more generic parallel programming model with a hierarchy of parallel threads, barrier synchronization and atomic operations to dispatch and manage highly parallel computing work. NVIDIA also released the CUDA C/C++ compiler (NVCC), libraries and runtime software to enable programmers to access the new data-parallel computation model and to develop applications.

5.2.1 CUDA Architecture

The architecture of a typical CUDA capable NVIDIA GPU is presented in this section. The GPU is organized into an array of highly threaded streaming multi-processors (SM). Each SM has a number of *streaming processors (SP)* that share control logic and instructions cache. Each GPU has up to 4GB of graphics double data rate (GDDR) RAM, referred as global memory which is essentially the frame buffer memory. Normally, this involves video images and texture information for 3D rendering but for computing purposes it is a very high-bandwidth, off-chip memory but with more latency than typical system memory. But for massively parallel applications, the higher bandwidth counterbalances for the longer latency.
The G80 has 86.4GB/s of memory bandwidth plus an 8GB/s communication bandwidth with the CPU, 4GB/s in each direction. The GPU is connected via the PCIe whose bandwidth is comparable to the CPU front-side bus bandwidth to the system memory. The G80 chip has

128 SP (16 SMs with 8 SPs each) and each SP has a multiply add (MAD) unit and an additional multiply unit. Summarized, this is a theoretical total of over 500 gigaflops. In addition, each SM has special function units that perform special floating-point functions like SQRT as well as transcendental functions. With the subsequent next generation GT200, the number of streaming processor cores increases (240 SPs), the processor register size was doubled, hardware memory access coalescing was added to improve memory access efficiency and double precision floating point were supported to address the need of scientific and HPC applications. The GT200 has a peak performance of more than 1 teraflop. The most recent GPU is the GF100, called FERMI and implemented in the NVIDIA GeForce 400 series. With it, the double precision performance was improved, the memory ECC supported, a true L1/L2 cache hierarchy was implemented with a additional 384kB L2 cache, more shared memory was available by a 64kB splitted shared memory/L1 cache, faster context switching and faster atomic operations. The graphics cards are equipped with up to 16 SMs, each SM with 32 cores, each with 1024 registers and up to 48 threads per core. For more technical details see [44].

5.2.2 CUDA Programming Model

In the first place, the CUDA programming model distinguishes between host and device. The host is the CPU with the system memory, the device is the GPU with its associated memory. A CUDA program is a unified source code encompassing both host and device code. The NVIDIA C compiler (NVCC) separates them during the compilation process. The host code is straight C code, compiled with the standard C compiler and runs as an ordinary CPU process. The device code is written using ANSI C extended with keywords for labeling data-parallel functions, called kernels and their associated data structures. The device code is further compiled by the NVCC and executed on a GPU device. CUDA C is a language integration making it possible to call device code from host code. The execution starts with host execution. When a kernel function is launched, the execution is moved to the device where a large number of threads is generated to take advantage of enormous data parallelism. All the threads that are generated by a kernel during an invocation are collectively called a grid. CUDA C provides library routines for memory allocation on graphics card, data transfer (for example constants, texture arrays, ordinary data) from host memory to device memory and back, for error-checking and timing. For launching multiple copies of the kernel process on the GPU a special syntax is defined. These kernel functions generate a large number of threads to exploit data parallelism. A typical GPU program with explicit movement of data across the PCIe connection by the master process running on the CPU performs the steps, given in Algorithm 5.1.

5.2.2.1 Threads

Kernel functions specify the code to be executed by all threads during a parallel phase. When launching a kernel, it is executed as a grid of parallel threads. Each CUDA grid typically contains of thousands to million of lightweight GPU threads per kernel invocation. One way to hide latency is to create many threads to fully utilize the hardware. This often requires a large amount of data parallelism. Threads in a grid are organized into a two-level hierarchy: in the

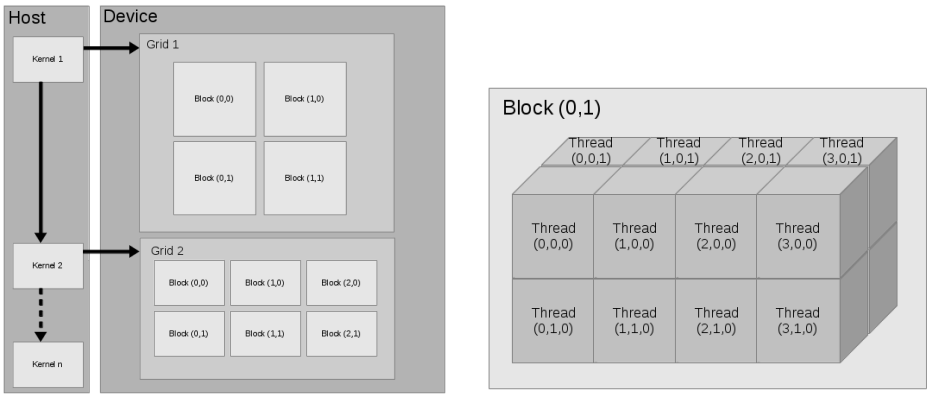
Algorithm 5.1 Typical GPU program

1: initialize graphics card
2: allocate memory in host and on devices
3: copy data from host to device memory
4: launch multiple copies of execution kernel on devices
5: copy data from device memory to host
6: repeat 3-5 as needed
7: de-allocate all memory and terminate.

first level, grids consist of one or more 2D blocks and in the second level, every block covers a 3D array of threads, see Figure 5.2. Every thread has unique coordinates distinguishing it from the others and to identify the appropriate portion of the data to process. The syntax for launching a kernel is

```
invoke_kernel<<<gridDim, blockDim>>>(args);
```

gridDim denotes the number of copies of the kernel, *blockDim* is the number of threads within each copy and *args* is a limited number of arguments, mainly pointers to arrays in graphics memory and some constants which get copied by value.



(a) Grid (b) Block

Figure 5.2: Kernel launch according to [37]

Threads within the same block are allowed to synchronize by the use of a barrier synchronization function, which must be executed by all threads in a block. No thread is allowed to proceed beyond this barrier until the rest has reached it. Applied improperly, this can end up in a deadlock. All threads within a block should execute in close time proximity with each other to avoid excessively long waiting times. The CUDA runtime systems satisfy this constraint by assigning execution resources to all threads in a block as unit. CUDA programmers can assume that these threads take very few cycles to generate and schedule due to efficient hardware support. When a thread of a block is assigned to an execution resource, all other threads in the same block are also assigned to the same resource. All threads within one copy can access local shared memory but cannot see what the other copies are doing, even if they are on the same SM. This leads to a trade off: by not allowing threads in different blocks to perform barrier synchronization with each other, the CUDA runtime can execute

blocks in any order relative to each other because none of them must wait for each other. This is called *transparent scalability*. There are no guarantees on the order in which the copies will be executed. When a copy of the kernel is started on a SM it is executed by a number of threads, each of which has the following information: some variables passed as arguments, pointers to arrays in device memory, global constants in device memory, shared memory and private registers/local variables and some special variables. Up to 8 blocks can be assigned to each SM in the GT200 design as long as there are enough resources to satisfy the demand of all the blocks.
The blocks on each SM are further divided and executed in groups of 32 so-called *warps*. Execution alternates between active warps and with warps becoming temporarily inactive when waiting for data. For each thread, one operation completes before the next starts. This avoids the complexity of pipeline overlaps which can limit the performance of modern processors.
The kernel code is written from the point of view of a single thread and the knowledge of the range of each variable. The CUDA SM warp scheduler decides in each clock cycle which warp to execute next, choosing from those neither waiting for data from the device memory (*memory latency*) nor for completion of earlier instructions (*pipeline delay*). The key to high performance is to keep a lot of active threads and warps around to hide latency. On the other side, because there is no context switch as each thread has its own register, the number of active threads is limited. But the selection of ready warps waiting for execution does not introduce any idle time into the execution timeline which is referred to as *zero-overhead thread scheduling*. Warps will be padded if they are not a multiple of the warpsize. For multiple dimensions of threads, dimensions will be projected into a linear order before partitioning into warps, see Figure 5.3.

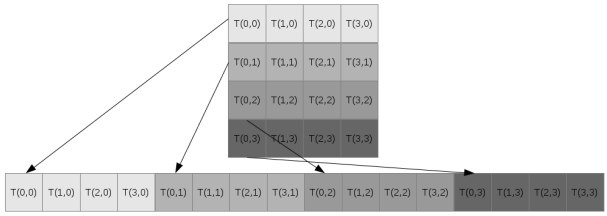


Figure 5.3: Linear order

The treatment of if-then-else branches is very special: threads are executed in warps of 32, with all threads in the warp executing the same instruction at the same time. If different threads in a warp need to do different things, the so-called *warp divergence* occurs. Then all the threads execute both (or even more) conditional branches, so the cost execution is the sum of both branches, which is a potentially large loss of performance. If a particular warp goes one way, then that is all that is executed. As each warp is treated separately, it is not of interest what is happening with other warps. If each warp only goes one way it is very efficient. In the worst case, one effectively loses factor 32 in performance if one thread needs an expensive branch, while the rest is idling. Warp divergence can have a very big impact on performance, see the NVIDIA Best Practice Guide [42].

5.2.2.2 Memories

The key challenge in modern computer architecture is the following: fast computation does not bring impact if data cannot be moved in and out fast enough. For big applications, there is a need for lots of memory. Very fast memory is also very expensive, so modern computers provide a hierarchical memory design. In Figure 5.4 the different memories on a TESLA GPUs beside the *global memory* which is implemented with dynamic random access memory (DRAM) having long access latencies (hundreds of clock cycles) are presented. Global arrays are held in the large device memory (DRAM), allocated by the host. The pointers are held by the host and passed into kernels and continue to exist until freed by host code. Since blocks are executed in arbitrary order, if one block modifies an array element, no other block should read or write the same element. A global array can also be created by declarations with global scope within kernel code.

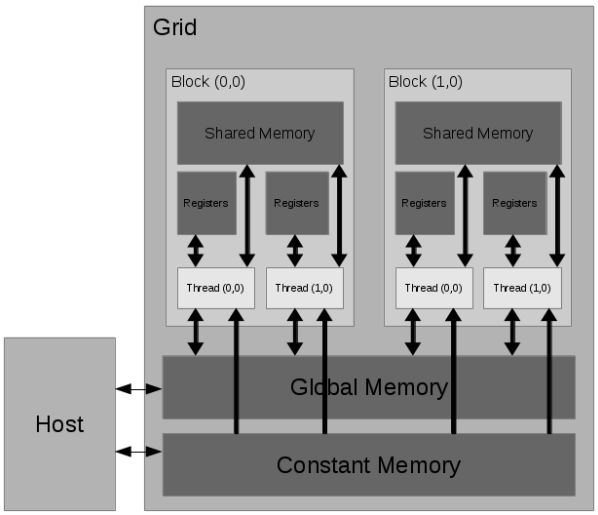


Figure 5.4: Memory hierarchy according to [37]

The *constant memory* is a global memory only to be addressed by the host, supporting long latency, high-bandwidth and read-only access by the device when all threads simultaneously access the same location. Constant variables cannot be modified by kernels. There is 64kB of constant memory and up to 48kB of cache for every SM. Registers and shared memory are on-chip memory, the variables residing in these types of memory can be accessed at a very high speed in a highly parallel manner. *Registers* are allocated to individual threads: each thread can only access its own registers. FERMI exemplary provides 32K 32bit registers per SM. If an application needs more registers, they spill over in device memory, so the application suffers from the latency and bandwidth implications of using device memory. *Shared memory* is allocated to thread blocks: all threads in a block can access variables in the shared memory location allocated to the block. This is an efficient method for threads to cooperate by sharing their input data and the intermediate results of their work. Accessing shared memory is extremely fast and highly parallel. Beside this, there are several further benefits: it is essential for operations requiring communication between threads, the improvement of data re-use, an alternative to local arrays in the device memory and to reduces the use of registers when a variable has the same value for all threads. But if a thread block has more than one warp, it is

not predetermined when each warp will execute its instructions. Consequently, almost always thread synchronization is needed to ensure the correct use of shared memory.
A wide bus, as mentioned above, is the only way to get a high bandwidth to the slower main memory. The cache line is the basic unit of data transfer with a typical size of 128 bytes. For FERMI, a cache line contains of a number of 32 floats or 16 doubles. There exist a 384bit memory bus from the device memory to the L2 cache with a bandwidth up to 160GB/s and a unified 384kB L2 cache for all SMs. Each SM has 16kB or 48kB L1 cache (64kB is split 16/48 or 48/16 between L1 cache and shared memory). As there is no global cache coherency as in CPUs, one almost never has different blocks updating the same global array elements.
A memory coalescing technique is implemented: as the access to DRAM is a very slow process, modern DRAMs use a parallel process to increase their rate of data access. Each time, a location is accessed, many consecutive locations that include the requested location are accessed. If an application can make use of data from multiple, consecutive locations before moving on to other locations, the DRAM can supply the data at a much higher rate than if a truly random sequence of locations was accessed. By organizing memory accesses of threads in favorable patterns, high global memory access efficiency is achieved. The most favorable access pattern is achieved when the same instructions for all threads in a warp accesses consecutive global memory locations. In this case, the hardware coalesces all of these access into a consolidated access to consecutive DRAM locations.

5.3 Implementation of the CSMG - Details and Remarks

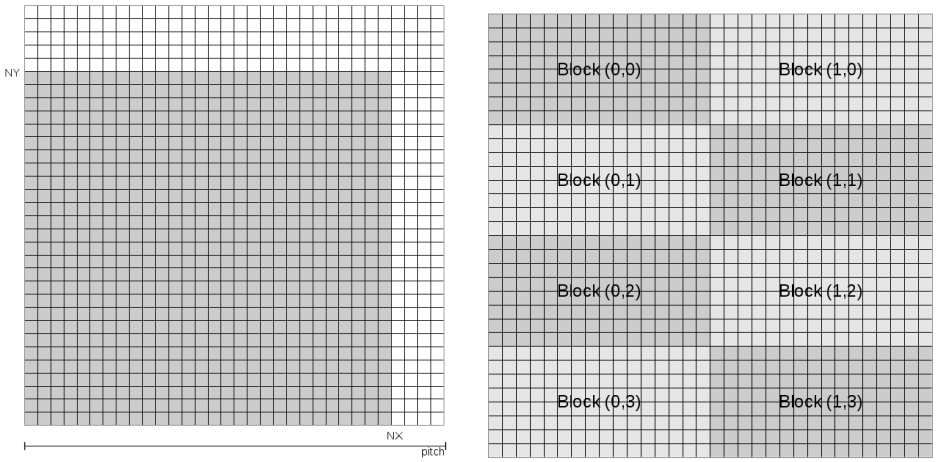
In this section, some of the implementation details of the CSMG on the GPU is described. In the Best Practice Guides [42], NVIDIA helps the programmer to obtain the best performance by giving specific recommendations with different priorities:

- find ways to parallelize sequential code
- minimize data transfer between host and device, even if it means to run some kernels on the device that do not show performance gains when compared with running on the host CPU
- adjust the kernel launch configuration to maximize device utilization
- ensure that global memory accesses are coalesced
- replace global memory accesses with shared memory accesses whenever possible
- avoid different execution paths within the same warp.

5.3.1 Global Memory Utilization on the GPU

For minimizing the data transfer between host and device, the level of discretization is chosen in a way that the entire problem fits in the DRAM of the GPU. In the beginning, all data is transferred to the GPU, all computation is done there and in the end, the results are copied back. The multigrid method is a recursive algorithm and temporary data structures are created on each level. On coarser levels, it is impossible to fully utilize the GPU and the CPU would show better performance. But as data transfer from host to device is expensive, all

computations are performed on the GPU. For the use of the already mentioned cache line, the memory is allocated on the GPU with the routine *cudaMallocPitch*, where the data are padded in x_1-direction to a multiple of 64, so each row starts at the beginning of a cache line, see Figure 5.5(a).



(a) cudaMallocPitch (b) Partitioning in tiles

Figure 5.5: Padding and Partitioning

5.3.2 Threads and Memories

For the two-dimensional implementation of the CSMG, the entire data is partitioned into subsets called *tiles* which can be treated independently of each other, see Figure 5.5(b). The size of each tile is determined by its memory demand - the data required for computing each tile should fit in the 48kB shared memory. Each block of threads is responsible for a tile, and each thread inside these blocks is responsible for one grid point. As the global memory is large but slow, whereas the shared memory is small but fast, the use of the shared memory is favorable if data are requested multiple times, for example when computing a matrix-vector product. In Figure 5.6, the dark-colored area is the corresponding tile and the light-colored cells are the halos (or ghost cells).

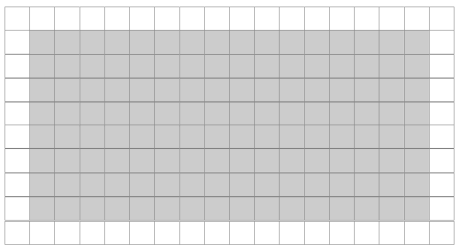


Figure 5.6: Shared memory

Therefore, each thread block copies the data from the global memory into the shared memory, using coalesced transfer as much as possible, synchronizes to ensure that previous steps are completed, computes new values and writes them back into the global memory. A typical

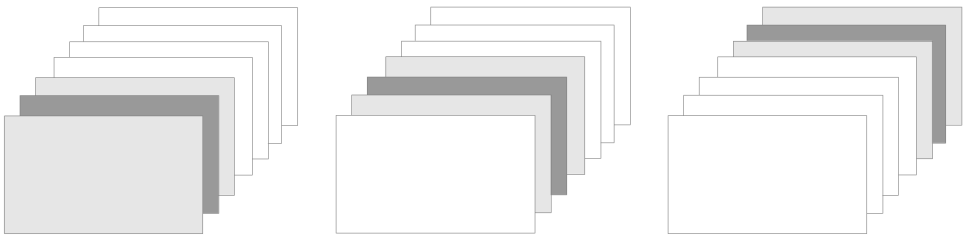
GPU programming code for a two-dimensional problem partitioned in tiles is given below in Code 5.1. Each tile has the size BLOCKX×BLOCKY.

CODE 5.1.

```
__global__ void function(double *indata, int NX, int NY,
      int pitch, double *outdata)
{
int i,j;
//load data in shared memory
//-----------------------------------------------------------
//allocate shared memory
__shared__ double data[(BLOCKX+2)*(BLOCKY+2)];

//load halos
int k=threadIdx.x+threadIdx.y*BLOCKX;
int halo= k< 2*(BLOCKX+BLOCKY+2); //threads to load halos
if (halo)
{
  if (threadIdx.y<2) //load y-halos coalesced
  {
    i=threadIdx.x;
    j=threadIdx.y*(BLOCKY+1)-1;
  }
  else //load x-halos uncoalesced
  {
    i=(k%2)*(BLOCKX+1)-1;
    j=k/2-BLOCKX-1;
  }
//local index halo in shared memory
int indh=(i+1)+(BLOCKX+2)*(j+1);

i=i+blockIdx.x*BLOCKX;
j=j+blockIdx.y*BLOCKY;
int indh_g=i+j*pitch; //global index halo in global memory

halo= (i>=0) && (i<NX) && (j>=0) && (j<NY);
}

//load main block in shared memory
i=threadIdx.x;
j=threadIdx.y;
int ind=(i+1)+(j+1)*(BLOCKX+2); //local index in shared memory

i=i+blockIdx.x*BLOCKX;
j=j+blockIdx.y*BLOCKY;
int_g=i+j*pitch; //global index in global memory

int active=(i<NX) && (j<NY)

//load main block and halos in shared memory
if (active) data[ind]=indata[ind_g];
if (halo) data[indh]=indata[indh_g];

//-----------------------------------------------------------
//do computations
if (active}
{
```

```
  if (i==0||i==NX-1||j==0||j==NY-1) //boundaries
  {
    //do something
  }
  else
  {
    //do something
    double res=...
  }
  outdata[indg]=res; //back to global memory
}
__syncthreads(); //barrier syncronization within the block
}
```

As the GPU grid is allocated with *cudaMallocPitch* and the computational grid, i.e. the discretized domains may differ, the request *if(active)* is necessary. For loading the halos into the shared memory, it is assumed that each tile has more interior grid points than halos. Coalesced transfer is guaranteed if the grid size is a multiple of 64 in x_1-direction and every block starts at a multiple of 64. The first condition is fulfilled by allocating memory with the *cudaMallocPitch* routine, the second by choosing a block size of a multiple of 64. The interior point of each tile correspond to a set of complete cache lines, the halo points above and below are complete cache lines, too. The remaining problem are the halo points on the left and right side: each one requires the loading of an entire cache line. These procedure is serialized.
For the implementation of the three-dimensional version of the CSMG, certain problems arise: the shared memory is limited to 48kB for each SM, therefore the implementation has to work with sliding windows with 3 planes of data being held at each instant, see Figure 5.7.



(a) sliding windows (b) sliding windows (c) sliding windows

Figure 5.7: Sliding windows

The (x_1, x_2) plane is cut up into tiles in the same way as the 2D application and each thread does an entire line in x_3-direction. Three planes are sufficient because of the 7-point stencil where only one neighboring grid point in x_3-direction is needed. The key steps in this code are given in Algorithm 5.2.

Algorithm 5.2 Sliding windows

1: load in $k = 0$ x_3-plane, including x_1 and x_2 halos
2: loop over all x_3-planes

a) load $k + 1$ x_3-plane (overwriting $k - 2$ plane)

b) process k x_3-plane

c) store new k x_3-plane.

and exemplary given as code

CODE 5.2.

```
__shared__ double data[3*(BLOCKX+2)*(BLOCKY+2)];

for (int k=0;k<NZ;k++)
{
  if (active)
  {
    indg0=indg; //store current global index
    indg=indg+NY*pitch; //move down one k-plane in global ids
    data[ind-(BLOCKX+2)*(BLOCKY+2)]=data[ind];
    data[ind]=data[ind+(BLOCKX+2)*(BLOCKY+2)];
    if (k<NZ-1)
    {
      data[ind+(BLOCKX+2)*(BLOCKY+2)]=indata[ind_g];
    }
  }
  if (halo)
  {
    ... same procedure
  }
}
__syncthreads();
```

Each block requires certain resources: a number of threads, registers and the total amount of shared memory. This restrictions decides how many blocks can run simultaneously on each SM - up to a maximum of 8 blocks. According to compute capability 2.0, 1536 threads (i.e. 48 warps with 32 threads) can run at once on a SM - if the other limitations (32768 registers per SM, 48kB shared memory) are not reached. For the CSMG it turned out to achieve best performance with a block size of 32x4. By numerical tests, other thread block sizes cannot remarkably improve the performance.

5.3.3 Reduction

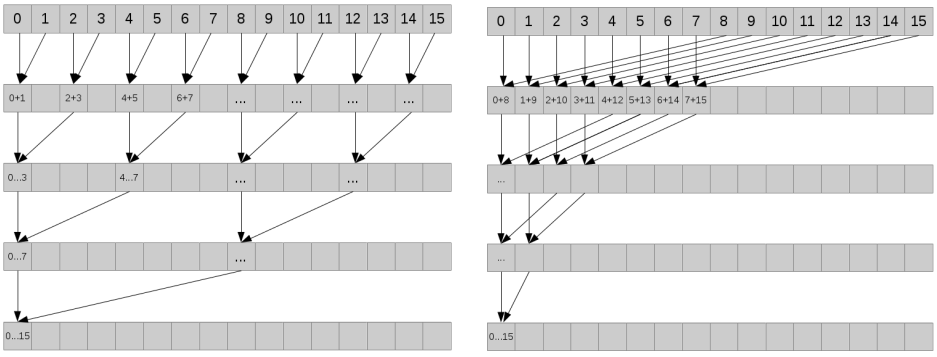
For computing the discrete L^2-norm of a vector as the termination condition for the multigrid method, in every cycle the sum of entries of a large array of values has to be determined. This occurs in many other cases, for example the vector dot product and is called *reduction*. Key requirements for a reduction operation are commutativity and associativity: the elements

can be rearranged and combined in any arbitrary order. The standard summation approach is sequential, see Code 5.3, adding one value at a time.

CODE 5.3.

```
double reduction(double *array, int N)
{
double sum=0.0;
for (int=0;i<N;i++)
  {
    sum+=array[i];
  }
return sum;
}
```

To parallelize this, a two-stage procedure is implemented: local and global reduction. Local reduction assumes each thread starts with one value, first adds the values within each thread block to form a partial sum together and then in the global reduction step it adds the partial sums from all the blocks. A detailed description of the algorithm can be found in [48, Chapter 5.3] . The parallel summation of N values works as follows: first sum them in pairs to get N/2 values and repeat the procedure until only one value remains. But on the GPU, the problem of warp divergence occurs as not all threads can be busy all of the time: in first phase N/2, in the second N/4, then N/8 and so on, see Figure 5.8(a). For efficiency reasons, one wants to make sure that each warp is either fully active or fully inactive, as far as possible, displayed in Figure 5.8(b). Another problem is that the threads need to access results produced by other threads. Hence access to global device memory would be very slow. To avoid this, shared memory is used, and synchronizing the data after each step to make sure that all previous operations have been completed. The memory is dynamically allocated when launching the kernel. The first thread outputs the final partial sum into a specific place for that block. When only one warp is working, synchronization is not needed anymore. It is guaranteed that previous operations by other threads in the same warp will have been completed because they are processed at the same time.



(a) bad kernel (b) good kernel

Figure 5.8: Warp divergence

The optimized code is given by

CODE 5.4.

```
__global__ void reduction(double *array, double *outdata)
{
extern __shared__ float temp[]; //dynamically allocated memory,
                                //size specified at kernel launch
int i=threadIdx.x;

temp[i]=array[i+blockIdx.x*blockDim.x];

for (int d=blockDim.x>>1; d>warpSize; d>>=1)
{ //>> is equivalent to dividing by 2
  __syncthreads();
  if (i<d) temp[i]+=temp[i+d];
}
__syncthreads();
if (i<warpsize) //only one warp
{
  for (int w=warpSize; d>0;d>>1)
  {
    if (i<d) temp[i]+=[tempi+d];
  }
}
//sum of local block reduction into global array
if (id==0) outdata[blockIdx.x]=temp[0];
}
```

This version puts the partial sum for each block in a different entry in a global array. Now there are at least two possibilities for global reduction: these partial sums can be transferred back to the host for the final summation or the local reduction kernel can be further applied.

5.3.4 Implementation of the Domain Decomposition Methods - Remarks

For the implementation of the domain decomposition methods, every subdomain is assigned to a single GPU. Each GPU is managed by a separate host thread by using *pthreads* for creating multiple threads and specifying with *CudaSetDevice* which GPU is used.
Beside the multigrid solves on the subdomains, setting up of the Schur complement matrix is processed on the GPU

$$S^{-1} = \bar{F}\Lambda^{-1}\bar{F}.$$

Since Λ^{-1} is a tri-blockdiagonal matrix, this results in a matrix-vector-matrix product. The entries of F are given explicitly and for this reason less memory transfer from CPU to GPU main memory is required. Otherwise a large amount of computations are processed, so the GPU is expected to use its full computational power and show a good performance. The matrix S^{-1} is stored in the CPU main memory and the matrix-vector products for determining the unknowns on the interfaces are performed on the CPU.

6 Numerical Results

This chapter is dedicated to the numerical results of the implementation of the collective smoothing multigrid method (CSMG) on a single domain in Section 6.1 as well as in Section 6.2 a comparison of the two proposed algorithms for the domain decomposition of the entire domain in many nonoverlapping subdomains, referred to as the recursive and the simultaneous approach. This chapter finishes in Section 6.3 with a proof of concept by the use of an example for large-scale optimization with more than 268 Mio. unknowns on 8 GPUs resp. CPUs in parallel.

6.1 Collective Smoothing Multigrid Method

In this section the implementation of the CSMG for the two- and three-dimensional test settings on one domain is considered. More precisely, the behavior of different choices of miscellaneous smoothers and the total number of smoothing steps on each grid for different problem sizes as well as the influence of the relaxation parameter α to the rate of convergence and the elapsed time on a GPU resp. CPU are investigated. The numerical results are presented for the in Chapter 2 introduced and analyzed academic optimal control problem with uniformly elliptic PDE-constraint

$$\min_{(y,u)} J(y,u) := \min_{(y,u)} \frac{1}{2}\|y - y_d\|_{L^2(\Omega)} + \frac{\alpha}{2}\|u\|_{L^2(\Omega)}$$

subject to

$$\begin{aligned} -\Delta y &= f + u \quad \text{in } \Omega \\ y &= 0 \qquad \text{on } \partial\Omega \end{aligned} \tag{6.1}$$

on the domain $\Omega = (0,1)^2$ in two and $\Omega = (0,1)^3$ in three dimensions. The mesh on the coarsest grid level ($k = 0$) consists of only one interior grid point where the resulting 2×2 system for the state and the adjoint variable is solved analytically. As target y_d, i.e. the state that is going to be tracked, without loss of generalization

$$y_d(x) = 1.0, \quad x \in \Omega \cup \partial\Omega,$$

is chosen, as displayed in Figure 6.1. This desired state can not be attained by any control since the zero boundary conditions can not fulfill the desired state on the boundaries. Nevertheless, the algorithms provide an optimal solution for the optimal control problem.

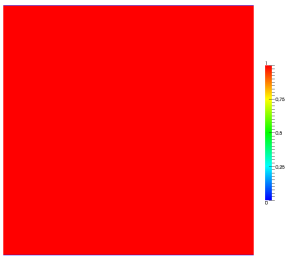


Figure 6.1: Target

The right-hand side f is given by

$$f(x) = 1.0, \quad x \in \Omega.$$

The relaxation parameter α varies in the range $[10^{-8}, 10^{-2}]$. Numerical tests have been done for two different smoothing iterations: the collective Jacobi relaxation with damping parameter ω (ω-JAC) and the collective red-black Gauss-Seidel (GS-RB). Several choices of the parameters, as the number of smoothing steps and different cycle schemes (V vs. W-cycle) as well as a comparison of single and double precision computation is presented. For the purpose of abbreviation let $V(\nu_1, \nu_2)$ denote a V-cycle with ν_1 pre-smoothing and ν_2 post-smoothing steps, analogously for $W(\nu_1, \nu_2)$. In the tables in the subsequent section, the convergence rate q, defined by

$$q := \sqrt[n]{\frac{d_n}{d_0}} := \sqrt[n]{\frac{\|\mathcal{A}_h w_h^n - \phi_h\|_0}{\|\mathcal{A}_h w_h^0 - \phi_h\|_0}}$$

where $\|\cdot\|_0$ denotes the previously in Chapter 3 defined discrete L^2-norm and n the number of iteration required to reduce the initial defect d_0 by a factor of $\epsilon = 10^{-8}$ for different grid sizes, given by the number N of grid points in x_1- and x_2- (x_3-)direction is presented. The starting value for all settings

$$w_h^0(x) = 0.0, \quad x \in \bar{\Omega}$$

was chosen.
Main purpose of the investigations in the successive sections is the elapsed time needed on a CPU compared to the time used on a GPU. *CPUt(s)* denotes the computation time in seconds on one CPU core, *GPUt(s)* the time required on the GPU in seconds, where the data transfer time between GPU and CPU global memory is included. The speed-up X is measured by the quotient

$$X := \text{speed-up}(CPU, GPU) := \frac{CPUt(s)}{GPUt(s)}.$$

The computations are performed on a desktop computer equipped with four INTEL XEON E5620@2.40 GHz CPU cores with hyper-threading, 12GB of RAM and a peak power requirement of 80 Watt. The 64bit operating system is LINUX Opensuse 11.3. All programs are compiled with the 4.4.1 version of the g++ compiler and the -03 option. The GPU is a NVIDIA GeForce GTX 470 graphics cards, providing 448 CUDA cores with a graphics clock frequency of 607MHz and a processor clock of 1215MHz. The standard memory configuration offers 1280MB GDDR5 memory and a memory interface width of 320-bit. This leads to

a theoretical memory bandwidth of 133.9 GB/sec. The minimum system power requirement is 550W and the GPU is connected to the motherboard via the PCI-E 2.0 x16 bus. The GPU uses the CUDA Driver CUDART with the driver version 4.2, the runtime 4.1 and exhibits compute capability 2.0. The NVCC compiler is of version 4.1.
In Table 6.1 and Table 6.2 information of the total number of unknowns is given, depending on the grid size $h = 1/N$ in each direction. For two dimensions, with $(N-1)$ interior grid points in x_1- and x_2-direction, it holds

N	#unknowns
16	450
32	1.922
64	7.938
128	32.258
256	130.050
512	522.242
1024	2.093.058
2048	8.380.418
4096	33.538.050

Table 6.1: #unknowns in 2D

and in three dimensions with $(N-1)$ interior grid points in x_1-, x_2- and x_3-direction the data

N	#unknowns
16	6.750
32	59.582
64	500.094
128	4.096.766
256	33.162.750

Table 6.2: #unknowns in 3D

are valid.

6.1.1 Two-dimensional Case: Collective Damped Jacobi Relaxation (ω-JAC)

The first numerical tests deal with the collective damped Jacobi relaxation in two space dimensions. Table 6.3 and Table 6.4 show the influence of the damping parameter ω for V(1,1) and V(2,2) with a fixed grid size $N = 4096$, resulting in a linear system with $\sim$33.5 Mio. unknowns, where the values of the relaxation parameter α vary. Purpose of these tests is to to quantify a range for the damping parameter where the best rate of convergence and the fastest elapsed time on both architectures, CPU and GPU, is achieved.

	0.6-JAC				0.7-JAC			
α	n	q	CPUt(s)	GPUt(s)	n	q	CPUt(s)	GPUt(s)
10^{-2}	28	0.5145	73.44	4.81	23	0.4473	60.44	4.02
10^{-4}	27	0.5009	70.89	4.66	23	0.4340	61.17	4.03
10^{-6}	26	0.4847	68.38	4.50	22	0.4179	56.64	3.87
10^{-8}	25	0.4662	64.88	4.28	20	0.3975	52.10	3.49
	0.8-JAC				0.9-JAC			
α	n	q	CPUt(s)	GPUt(s)	n	q	CPUt(s)	GPUt(s)
10^{-2}	20	0.3832	53.05	3.67	18	0.3492	48.35	3.23
10^{-4}	19	0.3706	49.87	3.40	17	0.3377	45.83	3.09
10^{-6}	18	0.3549	47.56	3.23	17	0.3372	44.89	3.07
10^{-8}	17	0.3359	44.35	3.01	17	0.3367	44.22	3.01

Table 6.3: ω-JAC, V(1,1)

	0.6-JAC				0.7-JAC			
α	n	q	CPUt(s)	GPUt(s)	n	q	CPUt(s)	GPUt(s)
10^{-2}	15	0.2766	51.19	3.61	13	0.2212	44.51	3.17
10^{-4}	14	0.2641	47.87	3.39	12	0.2073	40.91	2.95
10^{-6}	14	0.2497	47.69	3.39	12	0.1953	41.15	2.95
10^{-8}	13	0.2323	44.43	3.17	11	0.1796	37.78	2.73
	0.8-JAC				0.9-JAC			
α	n	q	CPUt(s)	GPUt(s)	n	q	CPUt(s)	GPUt(s)
10^{-2}	12	0.1899	40.94	2.95	11	0.1702	37.69	2.73
10^{-4}	11	0.1659	37.57	2.73	10	0.1449	34.44	2.51
10^{-6}	10	0.1598	34.20	2.51	10	0.1422	34.29	2.53
10^{-8}	10	0.1425	34.19	2.51	9	0.1273	30.68	2.29

Table 6.4: ω-JAC, V(2,2)

For $\omega = 1.0$, no convergence was observed. It turns out that the best value for the damping parameter ω for all choices of α is achieved when chosen in the range of $[0.8, 0.9]$.

6.1.2 Two-dimensional Case: Collective GS-RB vs. ω-JAC

In this subsection two different smoothing iterations for the two-dimensional test setting are compared: the collective damped 0.8-Jacobi relaxation and the collective red-black Gauss-Seidel iteration. For both of the smoothers the influence of the relaxation parameter α for various grid sizes is investigated. For the 0.8-JAC, V-cycles with 1 and 2 pre- and post-smoothing steps and for the GS-RB the V(1,1)-cycle is considered.

	$\alpha = 10^{-2}$					$\alpha = 10^{-4}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	17	0.3248	0.0023	0.0079	0.29	15	0.2864	0.0019	0.0070	0.27
32	18	0.3473	0.0057	0.0137	0.42	16	0.3089	0.0051	0.0122	0.42
64	18	0.3592	0.0138	0.0192	0.72	17	0.3302	0.0141	0.0185	0.76
128	19	0.3667	0.0459	0.0310	1.48	18	0.3453	0.0436	0.0296	1.47
256	19	0.3716	0.1658	0.0604	2.75	18	0.3543	0.1571	0.0561	2.80
512	19	0.3752	0.6967	0.1222	5.70	19	0.3608	0.6951	0.1227	5.67
1024	19	0.3781	3.0305	0.3002	10.09	19	0.3649	3.0125	0.2978	10.12
2048	20	0.3808	12.9276	0.9611	13.45	19	0.3680	12.2885	0.9119	13.48
4096	20	0.3823	51.3348	3.4838	14.74	19	0.3706	48.6316	3.3294	14.61
	$\alpha = 10^{-6}$					$\alpha = 10^{-8}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	11	0.1629	0.0015	0.0052	0.29	6	0.0297	0.0008	0.0030	0.27
32	14	0.2639	0.0044	0.0107	0.41	7	0.0586	0.0022	0.0059	0.37
64	16	0.3132	0.0132	0.0171	0.77	11	0.1867	0.0101	0.0121	0.83
128	17	0.3221	0.0410	0.0279	1.47	15	0.2866	0.0361	0.0247	1.46
256	17	0.3291	0.1493	0.0544	2.74	15	0.2906	0.1323	0.0472	2.80
512	18	0.3397	0.0676	0.1164	5.65	16	0.3029	0.5861	0.1044	5.61
1024	18	0.3467	2.8423	0.2850	9.97	17	0.3203	2.6990	0.2697	10.01
2048	18	0.3515	11.6369	0.8695	13.38	17	0.3299	10.9972	0.8263	13.31
4096	18	0.3549	46.1659	3.1705	14.56	17	0.3359	43.4050	3.0127	14.41

Table 6.5: 0.8-JAC, V(1,1)

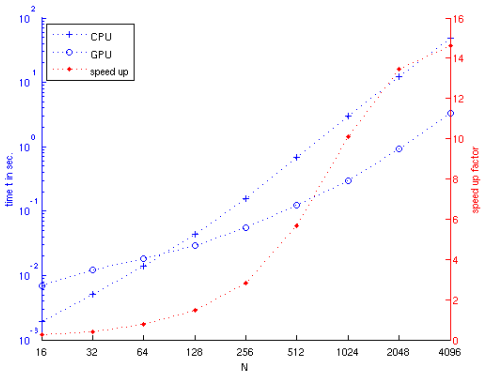
	$\alpha = 10^{-2}$					$\alpha = 10^{-4}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	11	0.1847	0.0017	0.0060	0.28	9	0.1142	0.0015	0.0049	0.31
32	11	0.1861	0.0045	0.0094	0.48	10	0.1406	0.0040	0.0086	0.47
64	11	0.1867	0.0114	0.0135	0.84	10	0.1512	0.0109	0.0121	0.90
128	11	0.1871	0.0350	0.0203	1.72	10	0.1568	0.0314	0.0187	1.68
256	11	0.1873	0.1245	0.0397	3.14	11	0.1610	0.1250	0.0412	3.03
512	12	0.1895	0.5651	0.0950	5.95	11	0.1631	0.5200	0.0866	6.00
1024	12	0.1896	2.4639	0.2441	10.09	11	0.1644	2.2726	0.2260	10.06
2048	12	0.1898	10.1657	0.7921	12.83	11	0.1653	9.2432	0.7311	12.64
4096	12	0.1899	40.2919	2.9440	13.69	11	0.1659	37.2237	2.7261	13.65
	$\alpha = 10^{-6}$					$\alpha = 10^{-8}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	6	0.0414	0.0010	0.0036	0.28	3	0.0009	0.0005	0.0019	0.26
32	9	0.1078	0.0036	0.0078	0.46	4	0.0044	0.0017	0.0039	0.44
64	10	0.1378	0.0076	0.0121	0.63	7	0.0663	0.0053	0.0086	0.62
128	10	0.1445	0.0322	0.0186	1.73	8	0.0998	0.0234	0.0152	1.54
256	10	0.1475	0.1177	0.0363	3.24	9	0.1179	0.1028	0.0331	3.11
512	10	0.1504	0.4727	0.0798	5.92	9	0.1269	0.4240	0.0724	5.86
1024	10	0.1526	2.0720	0.2074	9.99	10	0.1357	20.687	0.2097	9.87
2048	10	0.1540	8.4805	0.6686	12.68	10	0.1340	8.4086	0.6690	12.57
4096	10	0.1550	33.6477	2.5094	13.41	10	0.1425	33.7951	2.5068	13.48

Table 6.6: 0.8-JAC, V(2,2)

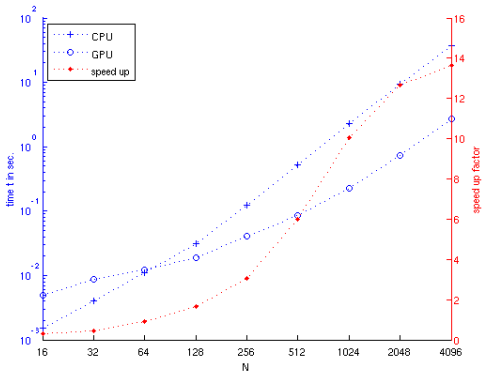
	$\alpha = 10^{-2}$					$\alpha = 10^{-4}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	9	0.1198	0.0008	0.0068	0.12	9	0.0933	0.0014	0.0044	0.32
32	9	0.1239	0.0030	0.0099	0.30	9	0.1183	0.0038	0.0071	0.54
64	9	0.1250	0.0059	0.0130	0.45	9	0.1211	0.0094	0.0132	0.71
128	9	0.1254	0.0253	0.0189	1.34	9	0.1218	0.0245	0.0188	1.30
256	9	0.1256	0.0878	0.0341	2.57	9	0.1219	0.0884	0.0340	2.60
512	9	0.1257	0.3689	0.0630	5.35	9	0.1220	0.3699	0.0691	5.35
1024	9	0.1258	1.5864	0.1706	9.30	9	0.1221	1.5819	0.1698	9.32
2048	9	0.1259	6.5701	0.5510	11.92	9	0.1222	6.5583	0.5519	11.88
4096	9	0.1259	26.3974	2.0908	12.63	9	0.1222	26.3559	2.0931	12.59
	$\alpha = 10^{-6}$					$\alpha = 10^{-8}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	8	0.0930	0.0011	0.0060	0.18	3	0.0006	0.0003	0.0018	0.17
32	7	0.0666	0.0015	0.0057	0.26	6	0.0256	0.0020	0.0068	0.29
64	8	0.0868	0.0053	0.0090	0.59	9	0.1786	0.0062	0.0100	0.62
128	8	0.0941	0.0225	0.0168	1.34	10	0.1482	0.0277	0.0208	1.33
256	8	0.0961	0.0780	0.0304	2.57	9	0.1279	0.0877	0.0340	2.58
512	8	0.0969	0.3287	0.0620	5.30	9	0.1211	0.3684	0.0692	5.32
1024	9	0.0976	1.4103	0.1530	9.22	9	0.1200	1.5827	0.1699	9.32
2048	8	0.0983	5.8517	0.4996	11.71	9	0.1198	6.5635	0.5477	11.98
4096	8	0.0980	23.4719	1.7778	13.20	9	0.1198	26.2260	1.9663	13.34

Table 6.7: GS-RB, V(1,1)

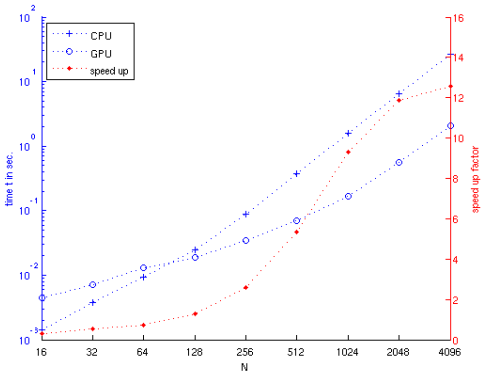
The tables 6.5, 6.6 and 6.7 show the results of the numerical experiments. They state a faster convergence for the red-black Gauss-Seidel compared to the Jacobi relaxation relating to the rate of convergence q and the computing time. The GPU version performs better for large problems ($N \geq 128$), the CPU version is faster for smaller grid sizes ($N < 128$). One reason is that on smaller grids the GPU cannot exploit its full computational resources (and power) and some of the streamprocessors idle. Furthermore, the number of iterations does not depend on the grid size which confirms optimal complexity. Comparing the values in Table 6.5 and in Table 6.6, the number of iterations decay if the number of smoothing steps is increased for the $\omega-$JAC. Moreover the number of iterations is robust in the regularization parameter α.
In the subsequent Figure 6.2, the elapsed time t for the CPU and the GPU implementation are displayed in blue color in a double logarithmic plot. The red scale gives information about the resulting speed-up factor.



(a) 0.8-JAC, V(1,1)



(b) 0.8-JAC, V(2,2)



(c) GS-RB, V(1,1)

Figure 6.2: Elapsed time and speed-up for $\alpha = 10^{-4}$

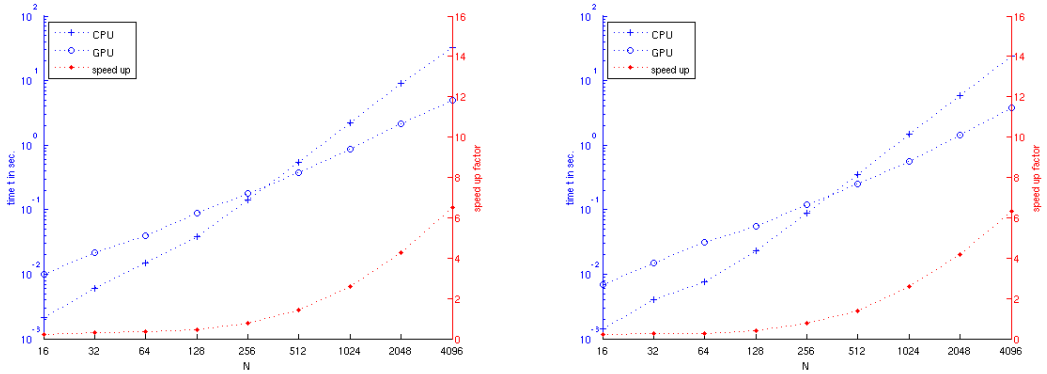
6.1.3 Two-dimensional Case: V- vs. W-cycle

In this subsection numerical tests for the W-cycle for both smoothing iterations are performed for different grid sizes $N \times N$ and constant relaxation parameter $\alpha = 10^{-4}$. The results for both smoothers are compared to the performance for the V-cycle tests.

	0.8-JAC W(2,2)					GS-RB W(1,1)				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	9	0.1047	0.0021	0.0099	0.21	7	0.0643	0.0014	0.0069	0.20
32	9	0.1052	0.0061	0.0215	0.28	7	0.0514	0.0040	0.0149	0.27
64	8	0.0996	0.0148	0.0398	0.37	7	0.0495	0.0076	0.0312	0.24
128	8	0.0955	0.0383	0.0884	0.43	6	0.0452	0.0229	0.0556	0.41
256	8	0.0915	0.1394	0.1805	0.77	6	0.0425	0.0888	0.1188	0.75
512	8	0.0876	0.5357	0.3720	1.44	6	0.0401	0.3471	0.2508	1.38
1024	8	0.0839	2.2264	0.8644	2.58	6	0.0378	1.4620	0.5683	2.57
2048	8	0.0803	9.1468	2.1442	4.27	6	0.0357	5.9291	1.4228	4.17
4096	7	0.0718	32.3972	4.9802	6.51	6	0.0369	23.9425	3.7880	6.32

Table 6.8: W - cycle 0.8-JAC and GS-RB, $\alpha = 10^{-4}$

Comparing the W(2,2)-cycle for the 0.8-JAC and the W(1,1)-cycle for GS-RB, the collective Gauss-Seidel provides a faster convergence relating the rate of convergence and the elapsed time. Similar behavior to the results for the the V-cycles occur. On smaller grids, the CPU performs better, on larger grids the GPU is faster. A small difference with regard to the the break-even point, where the GPU outperforms the CPU, occurs: this point is reached later (here: 512 instead of 128). One reason is that because of the design of the W-cycle more operations are performed on smaller grids compared to the V-cycle, so again some of the computational resources idle. Optimal complexity can be seen, as illustrated in Figure 6.3. A comparison of the V- vs. the W-cycle shows that for the W-cycle a better rate of convergence q is achieved resulting in less iterations n to attain the exit condition but with regard to the elpased time, both variants perform better with the V-cycle.



(a) 0.8-JAC, W(2,2) (b) GS-RB, W(1,1)

Figure 6.3: V vs. W cycle, $\alpha = 10^{-4}$

6.1.4 Two-dimensional Case: Smoothing Steps

In this subsection the focus is on the influence of the number of smoothing steps for both smoothing iterations onto the elapsed time when performing on CPU resp. GPU and the rate of convergence for a constant relaxation parameter $\alpha = 10^{-4}$, large numbers of grid points and both of V- and W-cycle.

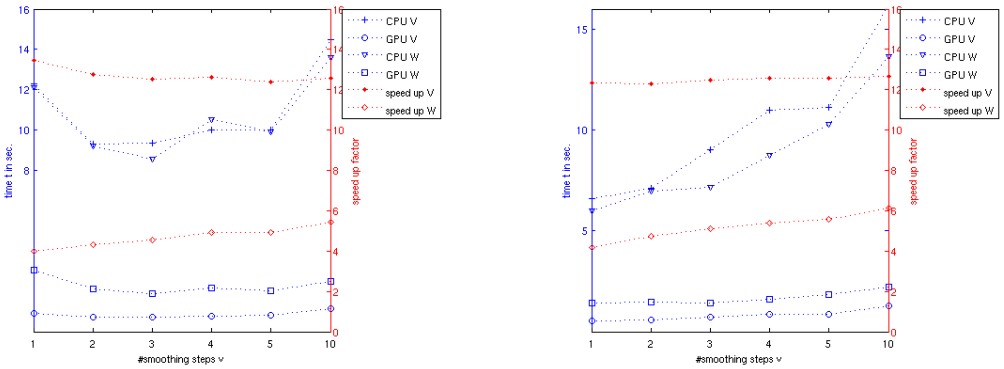
Increasing the number of smoothing steps on each grid leads to a faster convergence as the rate of convergence q goes down, but more amount of work that has to be done in each iteration step. The results for both cycle schemes are shown in Table 6.9 and Table 6.10. With respect to the elapsed time it turns out that for the 0.8-JAC 2 smoothing steps performs best, for the GS-RB a fast solution is achieved for only one step in the V-cycle and 1-2 steps for the W-cycle. Figure 6.4 shows the performance for both smoothing iterations and different cycle schemes for $N = 2048$, $\alpha = 10^{-4}$ and the resulting speed-up.

		0.8-JAC V(ν_1, ν_2)					0.8-JAC W(ν_1, ν_2)				
N	(ν_1, ν_2)	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
2048	(1,1)	19	0.3680	12.2900	0.9136	13.45	14	0.2621	12.1183	3.0477	3.98
	(2,2)	11	0.1653	9.2928	0.7297	12.74	8	0.0803	9.1947	2.1195	4.34
	(3,3)	9	0.1090	9.3467	0.7459	12.53	6	0.0391	8.5569	1.8814	4.55
	(4,4)	8	0.0853	9.9914	0.7914	12.63	6	0.0391	10.5265	2.1465	4.90
	(5,5)	7	0.0707	9.9943	0.8071	12.38	5	0.0214	9.9029	2.0118	4.92
	(10,10)	6	0.0337	14.4860	1.1526	12.55	4	0.0097	13.6123	2.5061	5.43
4096	(1,1)	19	0.3706	48.6781	3.3525	14.52	16	0.2557	48.5073	7.8291	6.19
	(2,2)	11	0.1659	37.1634	2.7260	13.63	7	0.0718	32.4445	4.9975	6.49
	(3,3)	9	0.1090	37.6711	2.8338	13.29	6	0.0368	34.5971	5.1410	6.73
	(4,4)	8	0.0853	39.8808	3.0379	13.13	5	0.0244	34.5821	4.9933	6.93
	(5,5)	7	0.0707	40.5014	3.1228	12.97	5	0.0199	40.4574	5.6535	7.16
	(10,10)	6	0.0337	58.7324	4.5334	12.96	4	0.0087	55.1732	7.2518	7.61

Table 6.9: Smoothing steps, V-cycle vs. W-cycle, 0.8-JAC, $\alpha = 10^{-4}$

		GS-RB V(ν_1, ν_2)					GS-RB W(ν_1, ν_2)				
N	(ν_1, ν_2)	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
2048	(1,1)	9	0.1222	6.5889	0.5345	12.33	6	0.0357	5.9797	1.4348	4.17
	(2,2)	7	0.0489	7.1197	0.5786	12.31	5	0.0119	6.9696	1.4728	4.73
	(3,3)	7	0.0461	9.0309	0.7248	12.46	4	0.0074	7.1565	1.4062	5.09
	(4,4)	7	0.0433	10.9706	0.8737	12.56	4	0.0054	8.7561	1.6193	5.41
	(5,5)	6	0.0281	11.1368	0.8856	12.58	4	0.0045	10.2955	1.8409	5.60
	(10,10)	5	0.0133	16.1799	1.2783	12.66	3	0.0019	13.6538	2.2261	6.13
4096	(1,1)	9	0.1222	26.3625	2.0090	13.12	6	0.0369	23.8388	3.8129	6.25
	(2,2)	7	0.0489	28.4877	2.2598	12.61	4	0.0092	22.5575	3.3639	6.71
	(3,3)	7	0.0461	36.6046	2.8843	12.69	4	0.0064	29.0756	4.0634	7.16
	(4,4)	7	0.0434	44.5760	3.5080	12.70	4	0.0049	35.6096	4.7802	7.45
	(5,5)	6	0.0282	44.8907	3.5945	12.49	4	0.0040	42.1396	5.4699	7.70
	(10,10)	5	0.0133	65.9557	5.2859	12.48	3	0.0017	56.2195	6.8444	8.21

Table 6.10: Smoothing steps, V-cycle vs. W-cycle, GS-RB, $\alpha = 10^{-4}$



(a) 0.8-JAC, V and W-cycle

(b) GS-RB, V and W-cycle

Figure 6.4: Smoothing steps for $N = 2048$, $\alpha = 10^{-4}$

6.1.5 Two-dimensional Case: Double vs. Single Precision

In this subsection the performance for single (SP) and double (DP) precision computation are compared for the red-black Gauss-Seidel smoothing iteration with $V(1,1)$. The relaxation parameter is set constant $\alpha = 10^{-9}$ and a reduction of the initial defect d_0 of a factor 10^{-5} is measured as for larger reduction factors no convergence was achieved due to the computational accuracy. These results were attained in part with the support of the HLRS in Stuttgart.

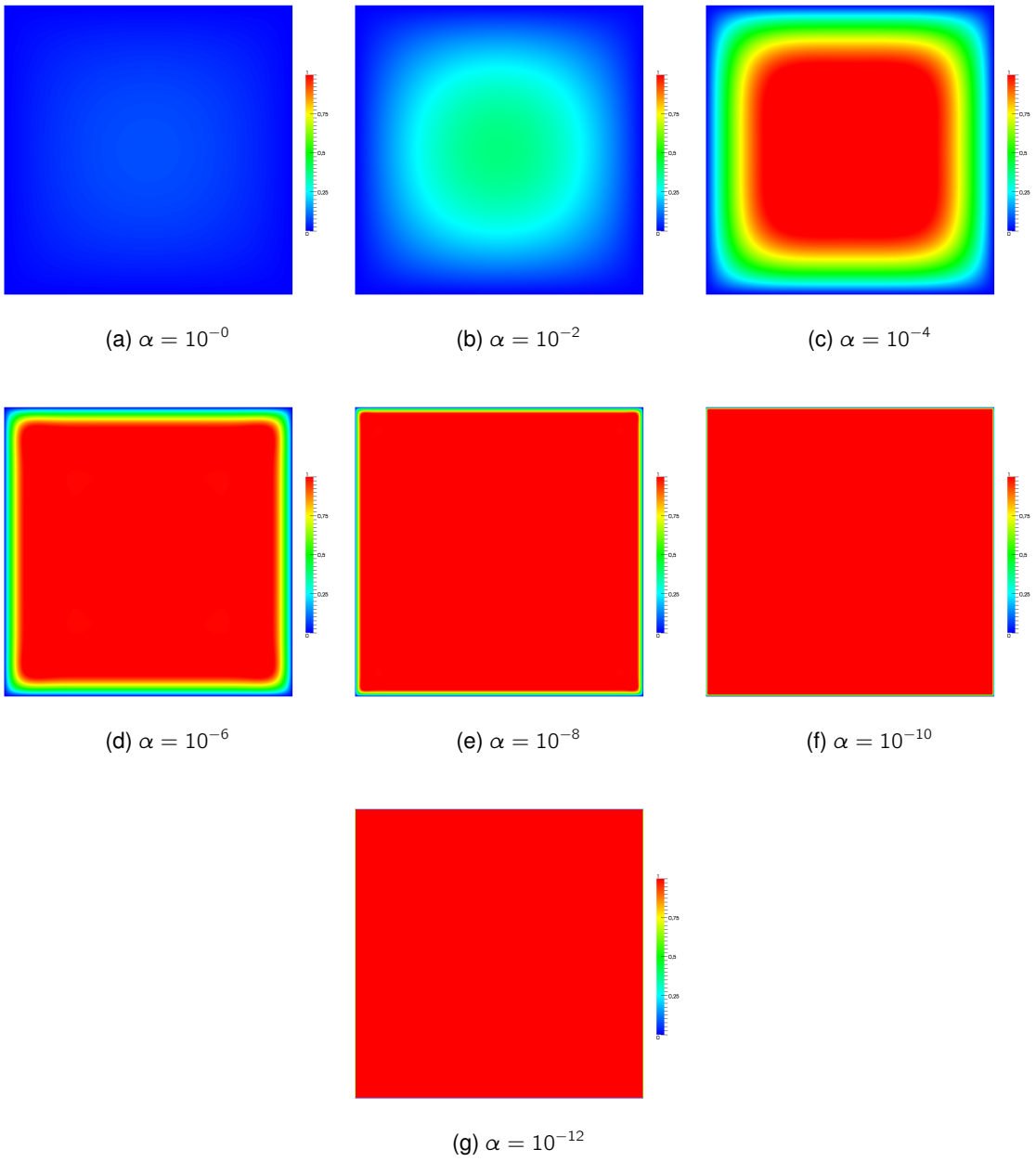
	GS-RB V(1,1) SP					GS-RB V(1,1) DP				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
512	6	0.1390	0.2253	0.0479	4.70	6	0.1389	0.2530	0.0562	4.50
1024	6	0.1370	0.9362	0.1096	8.54	6	0.1371	1.0828	0.1337	8.10
2048	6	0.1373	3.7456	0.3244	11.55	6	0.1366	4.4679	0.4009	11.14
4096	6	0.1425	15.0029	1.4375	10.44	6	0.1365	17.9305	1.4655	12.24

Table 6.11: Single vs. double precision, $\alpha = 10^{-9}$

For single precision the GPU and the CPU performs slightly better comparing to double precision.

6.1.6 Influence of the Relaxation parameter

In this subsection the numerical results are visualized for relaxation parameters α varying in the range $[10^0, 10^{-12}]$ and a grid size of 4096×4096 Figure 6.5 show the resulting optimal control, the target is given in Figure 6.1. For decreasing α the target function is tracked better, as expected. One can see the 'overhangs' near the boundary when the algorithm takes account of the discrepancy of the given target function with a constant value 1 on the boundary and the zero boundary conditions for the state, given by PDE-constraint in (6.1).



(a) $\alpha = 10^{-0}$ (b) $\alpha = 10^{-2}$ (c) $\alpha = 10^{-4}$

(d) $\alpha = 10^{-6}$ (e) $\alpha = 10^{-8}$ (f) $\alpha = 10^{-10}$

(g) $\alpha = 10^{-12}$

Figure 6.5: Optimal state depending on the relaxation parameter

6.1.7 Three-dimensional Case: ω-JAC vs. GS-RB

In this subsection the numerical results for the three-dimensional test setting are presented. In Table 6.12 and Table 6.13, the collective Jacobi relaxation with damping parameter $\omega = 0.8$ for the V- and W-cycle with 2 smoothing steps at each grid are compared.

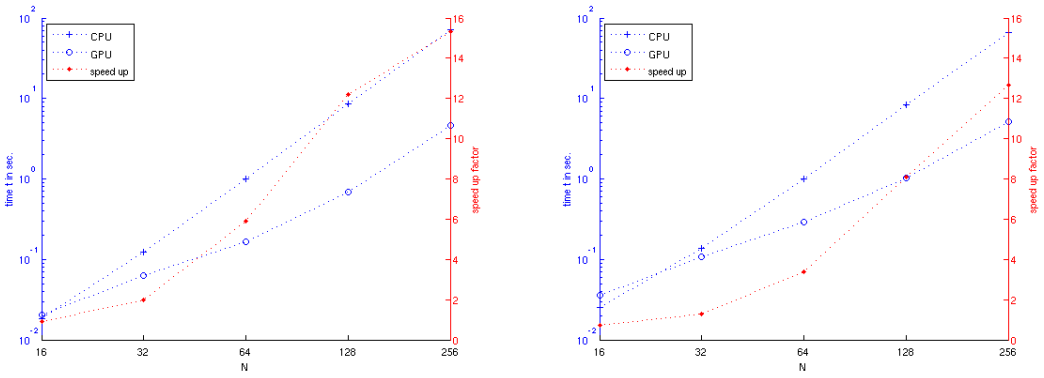
	$\alpha = 10^{-2}$					$\alpha = 10^{-4}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	15	0.2717	0.0262	0.0232	1.13	13	0.2400	0.0185	0.0203	0.91
32	15	0.2874	0.1393	0.0656	2.12	14	0.2466	0.1239	0.0628	1.97
64	16	0.2986	1.0590	0.1766	6.00	15	0.2705	0.9893	0.1671	5.92
128	16	0.3058	9.0564	0.7328	12.36	15	0.2864	8.4568	0.6928	12.21
256	16	0.3108	71.0136	4.3696	16.25	16	0.2957	71.1761	4.6401	15.34
	$\alpha = 10^{-6}$					$\alpha = 10^{-8}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	9	0.1118	0.0145	0.0144	1.01	3	0.0013	0.0044	0.0052	0.85
32	13	0.2293	0.1148	0.0571	2.01	5	0.0207	0.0481	0.0235	2.05
64	14	0.2486	0.9292	0.1570	5.92	10	0.1415	0.6807	0.1159	5.87
128	14	0.2596	7.9441	0.6473	12.27	13	0.2284	7.3751	0.6069	12.15
256	15	0.2696	66.8092	4.3801	15.25	13	0.2416	57.8037	3.8674	14.95

Table 6.12: 0.8-JAC, V(2,2)

	$\alpha = 10^{-2}$					$\alpha = 10^{-4}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	13	0.2370	0.0214	0.0355	0.60	13	0.2299	0.0250	0.0357	0.70
32	13	0.2410	0.1396	0.1102	1.27	13	0.2358	0.1376	0.1079	1.28
64	13	0.2382	1.0001	0.2915	3.43	13	0.2374	0.9919	0.2916	3.40
128	13	0.2333	8.2531	1.0264	8.04	13	0.2332	8.2795	1.0201	8.12
256	13	0.2279	65.6664	5.1859	12.66	13	0.2278	65.6017	5.1752	12.68
	$\alpha = 10^{-6}$					$\alpha = 10^{-8}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	9	0.1115	0.0154	0.0249	0.62	3	0.0013	0.0079	0.0091	0.87
32	13	0.2288	0.1392	0.1088	1.28	5	0.0207	0.0548	0.0437	1.25
64	12	0.1989	0.9164	0.2692	3.40	10	0.1406	0.7624	0.2268	3.36
128	13	0.2305	8.2882	1.0194	8.13	13	0.2238	8.3058	1.0221	8.13
256	13	0.2273	66.2453	5.2021	12.73	12	0.2129	61.7776	4.8507	12.74

Table 6.13: 0.8-JAC, W(2,2)

The insight is similar to the two-dimensional tests: the W-cycle provides a better rate of convergence than the V-cycle. Comparing the elapsed time, the CPU works better with W-cycle, the GPU with the V-cycle. The break-even point, where the GPU outperforms the CPU is for the V-cycle at a grid size 16, for the W-cycle at a grid size of 32. The CSMG provides optimal complexity and robustness with respect to the relaxation parameter α. Figure 6.6 shows the performance for the 3D case.



(a) 0.8-JAC, V(2,2), 3D (b) 0.8-JAC, W(2,2), 3D

Figure 6.6: 3D, 0.8-JAC, V vs. W cycle, $\alpha = 10^{-4}$

	$\alpha = 10^{-2}$					$\alpha = 10^{-4}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	13	0.2161	0.0168	0.0197	0.85	11	0.1722	0.0126	0.0168	0.75
32	13	0.2256	0.1046	0.0602	1.74	12	0.1899	0.0867	0.0538	1.61
64	13	0.2378	0.6995	0.1475	4.74	12	0.2032	0.6645	0.1360	4.89
128	14	0.2466	6.4941	0.6456	10.06	13	0.2226	5.9638	0.5987	9.96
256	14	0.2540	49.8577	4.0358	12.35	13	0.2356	45.7286	3.7773	12.11
	$\alpha = 10^{-6}$					$\alpha = 10^{-8}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	9	0.0968	0.0134	0.0149	0.90	3	0.0008	0.0051	0.0052	0.98
32	10	0.1580	0.0708	0.0442	1.60	6	0.0049	0.0443	0.0276	1.61
64	11	0.1762	0.5976	0.1239	4.82	9	0.1155	0.4937	0.1050	4.70
128	12	0.2009	5.5688	0.5592	9.96	11	0.1509	4.6276	0.4792	9.66
256	12	0.2090	42.4642	3.5228	12.05	11	0.1870	39.0655	3.2701	11.95

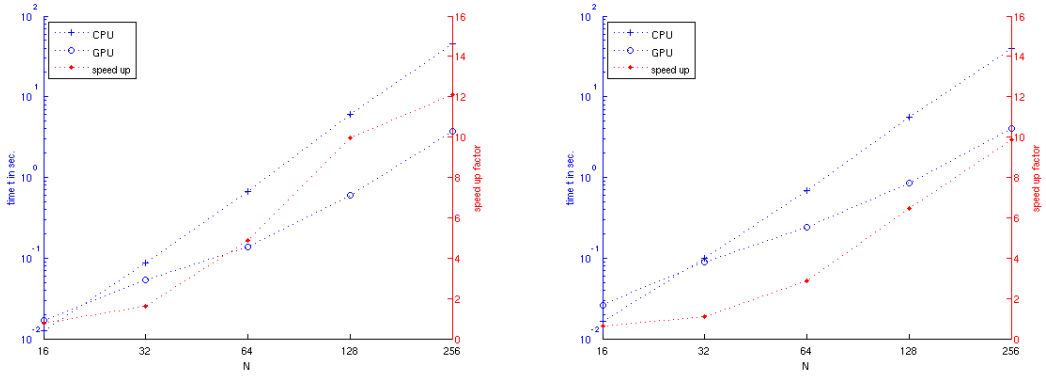
Table 6.14: GS-RB, V(1,1)

	$\alpha = 10^{-2}$					$\alpha = 10^{-4}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	11	0.1632	0.0180	0.0285	0.63	10	0.1580	0.0164	0.0262	0.63
32	11	0.1676	0.0908	0.0918	0.99	11	0.1622	0.0981	0.0881	1.11
64	11	0.1653	0.6630	0.2410	2.75	11	0.1648	0.6843	0.2387	2.87
128	11	0.1614	5.5358	0.8543	6.48	11	0.1613	5.5178	0.8524	6.47
256	10	0.1539	39.8232	3.9948	9.97	10	0.1539	39.4707	3.9979	9.87
	$\alpha = 10^{-6}$					$\alpha = 10^{-8}$				
N	n	q	CPUt(s)	GPUt(s)	X	n	q	CPUt(s)	GPUt(s)	X
16	9	0.1269	0.0146	0.0235	0.62	3	0.0008	0.0048	0.0083	0.58
32	11	0.1571	0.0952	0.0881	1.08	6	0.0462	0.0530	0.0490	1.08
64	9	0.1272	0.5439	0.1988	2.74	10	0.1466	0.6042	0.2199	2.75
128	10	0.1564	5.0151	0.7839	6.40	10	0.1498	5.0279	0.7834	6.42
256	10	0.1536	40.1878	4.0057	10.03	10	0.1428	39.5527	3.9906	9.91

Table 6.15: GS-RB, W(1,1)

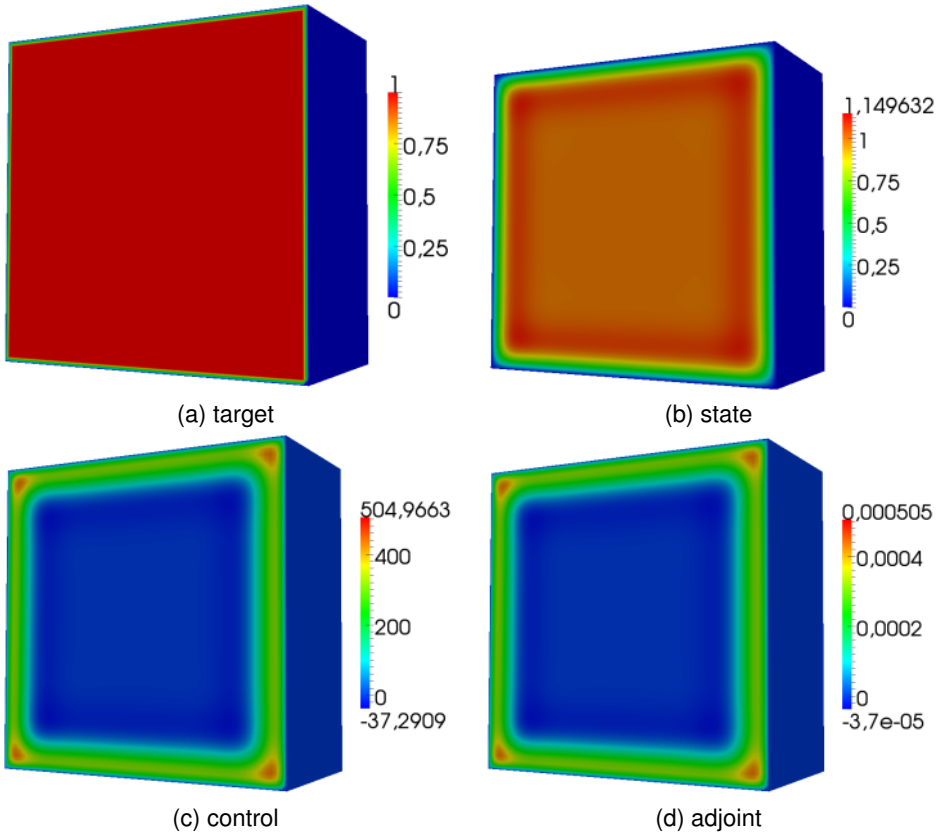
In Table 6.14 and Table 6.15, the collective Gauss-Seidel smoothing iteration for the V- and W-cycle with one smoothing steps at each grid are compared. The W-cycle provides a better

rate of convergence compared to the V-cycle. Relating the elapsed time, the CPU works better with the W-cycle, the GPU better with the V-cycle. Again the CSMG provides optimal complexity and robustness with respect to the relaxation parameter α, displayed in Figure 6.7. Comparing both smoothing iterations, the GS-RB performs better than the 0.8-JAC. A visualization of the numerical results is presented in Figure 6.8.



(a) GS-RB, V(1,1), 3D (b) GS-RB, W(1,1), 3D

Figure 6.7: 3D, GS-RB, V vs. W cycle, $\alpha = 10^{-4}$



(a) target (b) state

(c) control (d) adjoint

Figure 6.8: Optimal control in 3D, $\alpha = 10^{-6}$

6.2 Nonoverlapping Domain Decomposition: Many Subdomains

In this section the numerical results for the nonoverlapping domain decomposition are presented. The computations are processed on a compute server endowed with four INTEL XEON E5620@2.4GHz CPU cores with hyper-threading and 48GB of RAM. The operating system is LINUX Ubuntu 12.04 LTS. All programs are compiled with the 4.6.3 version of the g++-compiler and the -03 option. The GPU server is a TESLA M2050 rack equipped with 8 GPUs with a peak performance of 515 gigaflop double precision and 1030 single precision, 448 cores, 3 GB GDDR5 RAM, a theoretical memory bandwidth of 148 GB/sec. and the NVCC version 4.2.
The two in Chapter 4.5 proposed algorithms are matched: on the one side the recursive approach, on the other side the simultaneous approach. When comparing, in the first place one has to distinguish between *offline* and *online costs*. Offline costs is the work that can be computed in the preprocessing stage as these results can be reused several times within in the algorithm or for similar problems and e.g. do not depend on the right-hand side. In the case of the recursive approach, the setup of the Schur complements are considered as offline costs and for the simultaneous approach, the preliminary work for the BLOCKLU decomposition, in particular setting up the single blocks, is done in the preprocessing stage. The remaining work, like subdomain solves and matrix-vector products to determine the unknowns on the interfaces, are online costs. The algorithm for the BLOCKLU decomposition is given in [35].

6.2.1 Setup of the Schur Complement Matrix

Firstly the setup of the Schur complement matrix is considered. It consists of determining the diagonals of Λ resp. Λ^{-1} by solving a 1D-multigrid method for each grid point on the interface and computing the product

$$S^{-1} = \bar{F}\Lambda^{-1}\bar{F}. \tag{6.2}$$

The resulting Schur complement matrix has the size $2(N-1)\times 2(N-1)$. The computations of the entries of the diagonals of Λ are processed on the CPU. In Table 6.16 the times for setting up the Schur complement matrix by computing (6.2) on a CPU resp. GPU are given.

N	CPUt(s)	GPUt(s)	X
16	0.0005	0.0005	1.00
32	0.0047	0.0007	6.71
64	0.0396	0.0012	33.00
128	0.2974	0.0038	78.26
256	2.3611	0.0246	95.98
512	18.9842	0.1862	101.96
1024	151.4813	1.4586	103.85
2048	1210.2064	11.5949	104.37
4096	9643.7520	92.4783	104.28

Table 6.16: Setup time Schur Complement Matrix

The entries of the matrix F are given explicitly and do not have to be stored. As there is a lot of computation and less data transfer, the problem is ideally for the GPU and a large speed-up compared to a CPU is achieved.

6.2.2 Recursive vs. Simultaneous Solver

Numerical tests are performed for the nonoverlapping domain decomposition for the CSMG with the collective Gauss-Seidel smoother, V-cycle with one pre- and post-smoothing step. The relaxation parameter is chosen by $\alpha = 10^{-4}$ for all test settings and the solver on each subdomain terminates when the defect falls below a threshold of 10^{-12}. The domain is decomposed into uniform domains, i.e. all subdomains have the same size. According to the considerations in Section 4.5.3, a lot of effort can be saved as the Schur complement depends only on the sizes of the subdomains and once assembled, it can be reused for similar decompositions.

N	n	q	CPUt(s)	GPUt(s)	X
1024	10	0.1260	1.9982	0.3241	6.17
2048	10	0.1260	8.2183	0.8087	10.16
4096	10	0.1260	32.9251	2.6960	12.21
8192	9	0.1253	126.9764	nn	nn

Table 6.17: 1 SD, GS-RB V(1,1)

The results for the one subdomain case in Table 6.17 are already known by the preceding section. For $N = 8192$ the global memory of the GPU is too small and provides no result.

	offline		online		
N	getdiags	setup CAP	CPUt(s)	GPUt(s)	X
1024	19.1646	0.8766	2.7098	0.8006	3.38
2048	85.5034	6.9112	10.6614	1.9320	5.52
4096	396.0675	54.9281	57.7363	4.3695	13.21
8192	1749.8502	438.0821	200.8512	nn	nn

Table 6.18: 2 SD, GS-RB V(1,1)

For the two subdomain case both of the algorithms are equivalent due to the fact that only one Schur complement matrix is required. The computation are processed on two GPUs resp. two CPUs in parallel. For $N = 8192$ the GPU memory is still insufficient as both subdomains still have a grid size of 4096×8192. Comparing the online costs, the GPU is up to 13 times faster than the CPU. The offline costs are further splitted into determining the diagonals of Λ^{-1} and computing the matrix product $F\Lambda^{-1}F$. Figure 6.9 shows the optimal state for a domain decomposition into two subdomains on a domain with grid size 4096×2048.

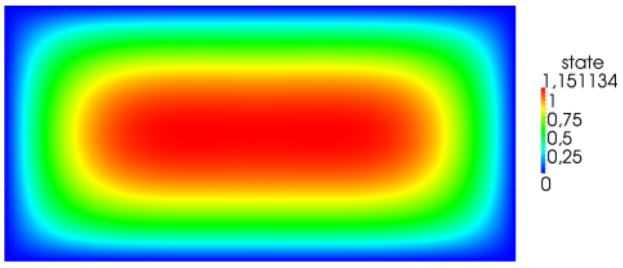


Figure 6.9: Domain decomposition into two subdomains

The next two tables present the numerical results for the recursive and simultaneous approach. Table 6.19 gives information about the online and offline for the recursive approach for a non-overlapping domain decomposition in four subdomains, parallelized on four GPUs resp. CPUs and Table 6.20 gives information for the simultaneous approach.

	offline		online		
N	getdiags	setup CAP	CPUt(s)	GPUt(s)	X
1024	27.4766	1.7556	3.5854	3.6798	0.97
2048	135.9447	13.8268	15.1114	5.5789	2.71
4096	612.3126	109.9152	92.1337	13.8628	6.65
8192	2590.4520	876.4320	292.0167	22.6234	12.91

Table 6.19: 4 SD, GS-RB V(1,1), recursive approach

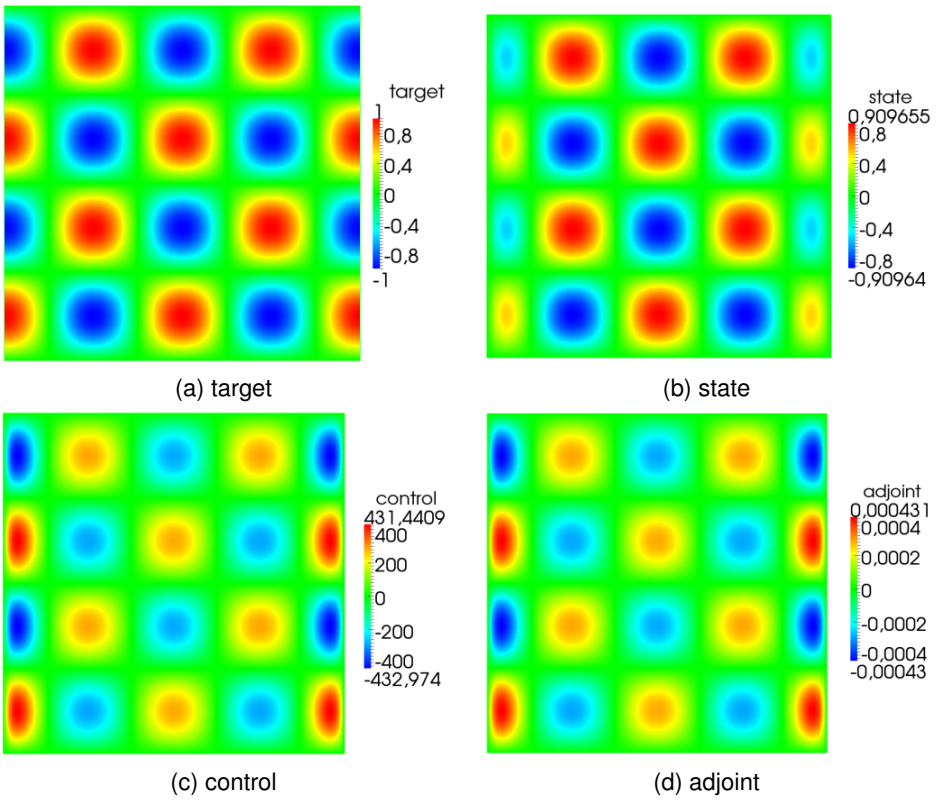
	offline		online		
N	getdiags	setup BLOCKLU	CPUt(s)	GPUt(s)	X
1024	8.8496	4.5236	1.7289	2.7407	0.63
2048	44.5075	80.3123	6.4247	4.5055	1.43
4096	208.7681	280.6874	37.4732	6.6819	5.61
8192	914.7330	2225.9949	143.6827	14.3912	10.03

Table 6.20: 4 SD, GS-RB V(1,1), simultaneous approach

Firstly comparing the *offline costs* of both approaches: here the computation of the diagonals for the Schur complement matrices and the setup of the Schur complement for the recursive approach as well as the setup of the BLOCKLU decomposition for the simultaneous approach are considered. Computing the diagonals happens faster for the simultaneous approach. Even if more Schur complement matrices have to be determined, the diagonals for the matrices on the offdiagonals in the interface system can be received during the process of computing the diagonals of Schur complement matrices on the main diagonal, see Chapter 4.5.2., and therefore some work can be saved. The setup of the BLOCKLU decomposition lasts longer than setting up the CAP matrices in the recursive approach. Summed both times up, the simultaneous approach is faster regarding to the offline costs for the considered problem sizes. Comparing the *online costs*, the simultaneous approach also leads to better results because of the less subdomain solves compared to the recursive approach. The main advantage of the recursive approach is the adaptivity to many subdomains. The extension is straightforward and for the the simultaneous approach, the solution of the Schur complement system with the BLOCKLU solver needs more work in the implementation and for computations. Comparing GPU to a CPU, a speed-up of 10x is attained.
An example for the decomposition into four subdomains is given in Figure 6.10. The two-dimensional target function is defined on $\overline{\Omega} = [0,1]^2$ by

$$y_d(x_1, x_2) = \sin(4\pi x_1)\cos(4\pi x_2)$$

and the relaxation parameter $\alpha = 10^{-6}$ was chosen. The grid has the size of 4096×4096.



(a) target (b) state

(c) control (d) adjoint

Figure 6.10: Domain decomposition into four subdomains

The following Table 6.21 gives information about the domain decomposition into eight subdomains for the recursive approach.

	offline		online		
N	getdiags	setup CAP	CPUt(s)	GPUt(s)	X
1024	32.2111	2.6261	7.5314	10.8579	0.69
2048	157.6899	20.7374	31.0829	21.0677	1.48
4096	729.2517	164.8501	241.7598	49.7219	4.86
8192	3103.5762	1314.9322	737.4715	77.8745	9.47

Table 6.21: 8 SD, GS-RB V(1,1), recursive approach

For scalability considerations recall in the considerations for the online costs in Section 4.5.3: For the recursive domain decomposition it holds

$$\frac{\text{effort}(n_{proc} \cdot V_0)}{n_{proc}} = n_{proc} \cdot const,$$

and for the simultaneous approach

$$\frac{\text{effort}(n_{proc} \cdot V_0)}{n_{proc}} = const.$$

These are theoretical values for an ideal scalable algorithm and not confirmed by the numerical results presented in Table 6.22 and Table 6.23. The label 'n SD, n GPU/CPU' means a

	1 SD, 1 GPU/CPU		2 SD, 2 GPU/CPU		4 SD, 4 GPU/CPU		8 SD, 8 GPU/CPU	
N	CPUt(s)	GPUt(s)	CPUt(s)	GPUt(s)	CPUt(s)	GPUt(s)	CPUt(s)	GPUt(s)
1024	1.9982	0.3241	2.7098	0.8006	3.5853	3.6798	7.5314	10.8579
2048	8.2183	0.8087	10.6614	1.9320	15.1114	5.5789	31.0829	21.0677
4096	32.9251	2.6960	57.7363	4.3695	92.1337	13.8628	241.7598	49.7219
8192	126.9764	nn	438.0821	nn	292.0167	22.6234	1314.9322	77.8745

Table 6.22: Scalability recursive approach

	1 SD, 1 GPU/CPU		2 SD, 2 GPU/CPU		4 SD, 4 GPU/CPU	
N	CPUt(s)	GPUt(s)	CPUt(s)	GPUt(s)	CPUt(s)	GPUt(s)
1024	1.9982	0.3241	2.7098	0.8006	1.7289	2.7407
2048	8.2183	0.8087	10.6614	1.9320	6.4247	4.5055
4096	32.9251	2.6960	57.7363	4.3695	37.4732	6.6819
8192	126.9764	nn	438.0821	nn	143.6827	14.3912

Table 6.23: Scalability simultaneous approach

nonoverlapping domain decomposition in n subdomains, performed on n GPUs resp. CPUs in parallel. There are several reasons: one point are problems arising of the design of these algorithms. Additional computation are processed in these algorithm, as matrix-vector product for solving the Schur complement system for the unknowns on the interfaces, which is not covered by the offline costs and included in the online cost. Data have to be copied from the device to the host and transferred back. Another point are technical difficulties. When using multiple processors communication overhead occurs. In particular, each GPU needs up to few second to get initialized as the driver has to be loaded every time when the first GPU kernel is launched.

6.3 Large-Scale Optimization

In this last section an example for large-scale optimization is presented. It is considered as a proof of concept. The domain Ω is decomposed into eight subdomains, each with 4096×4096 grid points, resulting in a linear system with ≈ 268 Mio. unknowns and solved with the recursive nonoverlapping domain decomposition method on eight GPUs resp. eight CPU parallel.

NX$\times$ NY	#	offline	CPUt(s)	GPUt(s)	X
4096$\times$32768	268.3Mio	5775.55	1499.66	118.21	12.7

Table 6.24: 8 SD, GS-RB V(1,1), recursive approach

Table 6.24 shows the numerical result, which ends in a achieved speed-up of $\approx 13\times$, comparing a GPU to a CPU.

7 Summary, Conclusions and Outlook

7.1 Summary

The aim of this work was the investigation of implementability and efficiency of an algorithm for solving optimal control problems on a new hardware architecture. For an academic test problem the collective smoothing multigrid method (CSMG) was realized on a commodity graphics card (GPU) and the performance in term of elapsed time compared to those on a recent CPU. For dealing with large problem size, new algorithms were designed and two different approaches considered: on the one hand, a recursive approach and on the other hand, a simultaneous approach. Both are based on a nonoverlapping domain decomposition of the entire domain into two subdomains, where a discrete approximation of the Steklov-Poincaré operator is derived by a Schur complement method. This so-called capacitance matrix is computed efficiently and inverted analytically. Numerical results show the performance of the CSMG for the one domain case and for both of the developed domain decomposition methods, comparing GPU and CPU. For large-scale optimization, an optimal control problem with $\approx$248 Mio. unknowns was solved by dividing the entire domain into 8 subdomains and processed on 8 GPUs/CPUs in parallel as a proof on concept.

7.2 Conclusions

Finally the conclusions are drawn, on the one side confirming well-known facts and on the other side giving new insights.

- The CSMG shows optimal convergence for the one domain case and robustness with respect to the relaxation parameter α.
- Regarding to the elapsed time on a GPU and a CPU, the CSMG with red-black Gauss-Seidel smoother works best with one smoothing step and the V-cycle. The CSMG with the damped Jacobi relaxation shows best performance with 2 smoothing steps and the V-cycle even though for W-cycles a better rate of convergence is achieved. Comparing both smoothing iterations, the CSMG with the red-black Gauss-Seidel as smoothing relaxation with one pre- and post-smoothing step is fastet.
- The GPU main memory size is limited. For large-scale problems, a domain decomposition is necessary to handle this problems.
- When using single precision, the algorithms do not provide a good solution (or even no solution at all) due to the computational accuracy. The most recent GPUs provide full IEEE support and the differences in term of the elapsed time do not show any noticeable difference when comparing double and single precision performance.

- Comparing GPU and CPU for the one domain case, the GPU is up to 13× faster than the CPU, regarding the elapsed time. For smaller grid sizes the CPU shows better performance. It turns out that the GPU performs best when the streamprocessors are fully utilized. This is the case when long latency access to data is minimized by using shared memory and having many threads ready to compute available. Transferring data from the CPU memory to the GPU memory is expensive. Even if some kernel on the device show no performance gain, they run on the GPU to avoid data transfer. Problems where there is a lot of computation and almost no data transfer are ideally for the GPU. This is, e.g., the case for the setup of the Schur complement matrix, where the GPU shows a speed-up of more than 100×.
- Comparing the both approaches for the nonoverlapping domain decomposition, the simultaneous approach performs better in both, online and offline costs. The BLOCKLU solver works well for the solution of the Schur complement system and needs less online costs as there are less subdomain solves. The advantage of the recursive approach is the simpler implementation and the adaptivity for partitions into more subdomains.
- Technical problems arise when using the GPU cluster: each GPU has a setup time where the driver has to be loaded - this could last up to a few seconds and takes sometimes longer than the invoked kernel.

7.3 Future work

The proposed algorithms and the implementation on a GPU cluster are considered as preliminary works for prospective multicore processors with several hundred cores. It will be challenging and interesting how the CSMG and the nonoverlapping domain decomposition methods perform on these architectures. Further investigations can test the performance on bigger GPU clusters and give more significant statements of the scalability of the proposed algorithms.
Every new generation of the GPUs and in particular the NVIDIA graphics cards needs slightly modification in the program code to achieve best performance. Recent NVIDIA GPUs provide new memory access possibilities. Therefore, auto tuning or preprocessor directives would be helpful to support the programmer.
CUDA programs only run on NVIDIA GPUs. An implementation in OpenCL to use graphics cards from other vendors or heterogenous computing can be tested in term of performance and develop programs that are platform-independent.
Some analytical work has to be done: various multigrid components can be tested numerically and analyzed by means of the local Fourier analysis. In the first place, the derivation of smoothing factors and error reduction factors for the collective damped Jacobi and the collective red-black Gauss Seidel iteration, which is not covered by this thesis. These factors have to be validated by numerical tests.
Another possible field of research is the analysis of the use of the Schur complement as preconditioner for more general differential operators on more general domain shapes.

List of Figures

List of Tables

Bibliography

[1] V. I. Agoshkov. Poincaré-Steklov Operators and Domain Decomposition Methods in Finite Dimensional Spaces. In R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, editors, *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, Philadelphia, PA, 1988. SIAM.

[2] H.W. Alt. *Lineare Funktionalanalysis: Eine anwendungsorientierte Einführung*. Springer-Lehrbuch Masterclass. Springer, 2006.

[3] E. Arian and S. Ta'asan. Multigrid one shot methods for optimal control problems: Infinite dimensional control. Technical report, Institute for Computer Applications in Science and Engineering (ICASE), 1994.

[4] J.-D. Benamou. A Domain Decomposition Method with Coupled Transmission Conditions for the Optimal Control of Systems Governed by Elliptic Partial Differential Equations. *SIAM Journal on Numerical Analysis*, 33(6):2401–2416, 1996.

[5] A. Borzì, K. Kunisch, and D. Y. Kwak. Accuracy and Convergence Properties of the Finite Difference Multigrid Solution of an Optimal Control Optimality System. *SIAM Journal on Control and Optimization*, 41:1477–1497, 2002.

[6] A. Borzì and V. Schulz. Multigrid Methods for PDE Optimization. *SIAM Review*, 51(2):361–395, 2009.

[7] A. Borzì and V. Schulz. *Computational Optimization of Systems Governed by Partial Differential Equations*. SIAM, 2012.

[8] J.-F. Bourgat, R. Glowinski, P. Le Tallec, and Ma. Vidrascu. Variational formulation and algorithm for trace operator in domain decomposition calculations. In *Domain decomposition methods (Los Angeles, CA, 1988)*, pages 3–16. SIAM, Philadelphia, PA, 1989.

[9] D. Braess. *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer, 1997.

[10] J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring I. *Mathematics of Computation*, 47(175):103–134, 1986.

[11] J. H. Bramble, J. E. Pasciak, and A. H. Schatz. An iterative method for elliptic problems on regions partitioned into substructures. *Mathematics of Computation*, 46(173):361–369, 1986.

[12] J.H. Bramble. *Multigrid methods*. Pitman research notes in mathematics series. Longman Scientific & Technical, 1993.

[13] J.H. Bramble, J.E. Pasciak, and J. Xu. The analysis of multigrid algorithms with non-nested spaces or non-inherited quadratic forms. *Mathematics of Computation*,

56(193):1–34, 1991.

[14] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[15] A. Brandt. Rigorous Quantitative Analysis of Multigrid, I: Constant Coefficients Two-Level Cycle with L2-Norm. *SIAM Journal on Numerical Analysis*, 31(6):1695–1730, 1994.

[16] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial (2nd ed.)*. SIAM, Philadelphia, PA, 2000.

[17] T. F. Chan. Analysis of preconditioners for domain decomposition. *SIAM Journal on Numerical Analysis*, 24(2):382–390, 1987.

[18] T. F. Chan and T. Y. Hou. Eigendecomposition of Domain Decomposition Interface Operators for Constant Coefficient Elliptic Problems. *SIAM Journal on Scientific and Statistical Computing*, 12:1471–1479, 1991.

[19] T. F. Chan and T. P. Mathew. Domain Decomposition Algorithms. In *Acta Numerica*, pages 61–143. Cambridge University Press, 1994.

[20] T. F. Chan and D. C. Resasco. A domain-decomposed fast Poisson solver on a rectangle. Technical Report /DCS/RR-409, Yale University, 1985.

[21] T. F. Chan and D. C. Resasco. An analysis of domain decomposition preconditioners on L-shaped and C-shaped regions. Technical Report /DCS/RR-534, Yale University, 1988.

[22] T. F. Chan and D. C. Resasco. A Framework for the Analysis and Construction of Domain Decomposition Preconditioners. In R. Glowinski, G. H. Golub, G. Meurant, and J. Périaux, editors, *Domain Decomposition Methods for Partial Differential Equations*, Philadelphia, PA, 1988. SIAM.

[23] M. Dobrowolski. *Angewandte Funktionalanalysis: Funktionalanalysis, Sobolev-Räume und Elliptische Differentialgleichungen*. Springer-Lehrbuch Masterclass. 2006.

[24] M. Dryja. A finite element-capacitance method for elliptic problems on regions partitioned into subregions. *Numerische Mathematik*, 44:153–168, 1984.

[25] M. Giles. Cuda Programming on NVIDIA GPUs. Course material, 2010.

[26] P. Grisvard. *Elliptic problems in nonsmooth domains*. Monographs and studies in mathematics. Pitman Advanced Publishing Program, 1985.

[27] C. Großmann and H.G. Roos. *Numerische Behandlung partieller Differentialgleichungen*. Teubner Studienbücher Mathematik. 2005.

[28] W. Hackbusch. On the regularity of difference schemes. Part II: Regularity estimates for linear and nonlinear problems. *Arkiv för Matematik*, 21:3–28, 1983.

[29] W. Hackbusch. *Multi-grid methods and applications*. Springer series in computational mathematics. 1985.

[30] W. Hackbusch. *Theorie und Numerik elliptischer Differentialgleichungen*. Teubner Studienbücher Mathematik. 1986.

[31] Wolfgang Hackbusch. On the regularity of difference schemes. *Arkiv för Matematik*, 19:71–95, 1981.

[32] Wolfgang Hackbusch. *Iterative Lösung grosser schwachbesetzter Gleichungssysteme*. B.G. Teubner, 1993.

[33] H. Heuser. *Funktionalanalysis*. Mathematische Leitfäden. Teubner, 1975.

[34] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich. *Optimization with PDE Constraints*. Mathematical modelling - theory and applications. Springer, 2009.

[35] E. Isaacson and H.B. Keller. *Analyse numerischer Verfahren*. Edition Leipzig, 1978.

[36] K. Karimi, N. G. Dickson, and F. Hamze. A Performance Comparison of CUDA and OpenCL. *Computing Research Repository*, abs/1005.2581, 2010.

[37] D. Kirk, W.W. Hwu, and W. Hwu. *Programming massively parallel processors: a hands-on approach*. Applications of GPU Computing Series. Morgan Kaufmann Publishers, 2010.

[38] J.L. Lions. *Optimal control of systems governed by partial differential equations*. Grundlehren der mathematischen Wissenschaften. Springer, 1971.

[39] P.-L. Lions. On the Schwarz alternating method. III: A variant for nonoverlapping subdomains. In Tony F. Chan, R. Glowinski, J. Périaux, and O. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, held in Houston, Texas, March 20-22, 1989*, Philadelphia, PA, 1990. SIAM.

[40] A. Meister. *Numerik linearer Gleichungssysteme: eine Einführung in moderne Verfahren*. Vieweg, 2005.

[41] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38:114–117, 1965.

[42] NVIDIA. CUDA C Best practices guide version 4.1. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf. last accessed: 08/30/2012.

[43] NVIDIA. NVIDIA CUDA C programming guide version 4.2. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf. last accessed: 08/30/2012.

[44] NVIDIA. NVIDIA's Next Generation CUDA Compute Architecture: Fermi. NVIDIA Whitepaper, 2009.

[45] A. Quarteroni and A. Valli. Theory and application of Steklov-Poincaré operators for boundary-value problems: the heterogeneous operator case. In R. Glowinski, Y. A. Kuznetsov, G. Meurant, J. Périaux, and O. B. Widlund, editors, *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, Philadelphia, PA, 1990. SIAM.

[46] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Oxford University Press, Oxford, UK, 1999.

[47] D. C. Resasco. *Domain decomposition algorithms for elliptic partial differential equations*. PhD thesis, Yale University, 1990.

[48] J. Sanders and E. Kandrot. *Cuda by Example: An Introduction to General-Purpose GPU Programming*. Pearson Education, 2010.

[49] J. Schoberl, R. Simon, and W. Zulehner. A Robust Multigrid Method for Elliptic Optimal Control Problems. *SIAM Journal on Numerical Analysis*, 49(4):1482–1503, 2011.

[50] H. A. Schwarz. Über einen Grenzübergang durch alternierendes Verfahren. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 15:272–286, 1870.

[51] S. Takacs and W. Zulehner. Convergence analysis of multigrid methods with collective point smoothers for optimal control problems. *Compututing and Visualization in Science*, 14(3):131–141, 2011.

[52] F. Tröltzsch. *Optimale Steuerung partieller Differentialgleichungen: Theorie, Verfahren und Anwendungen*. Vieweg, 2009.

[53] U. Trottenberg and A. Schuller. *Multigrid*. Academic Press, Inc., Orlando, FL, 2001.

[54] M. Vallejos and A. Borzí. Multigrid optimization methods for linear and bilinear elliptic optimal control problems. *Computing*, 82(1), 2008.

[55] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, 1962.