High-dimensional approximation for large-scale applications

Cumulative habilitation thesis
for the attainment of the degree
Dr. habil.
at the Faculty of Science
of the
University of Basel

handed in by
Dr. rer. nat. Peter Zaspel
from
Siegburg, Germany

Basel 2019

Abstract

Numerical approximation of high-dimensional problems based on models like partial differential equations (PDEs) is a key tool in research and development. High dimensionality in a given application can be based on an underlying physical problem of high dimensionality or it can be based on a high-dimensional parameter space being present in uncertainty quantification, inverse problems, optimization and machine learning. The objective of this cumulative habilitation thesis is to focus on the latter case and to introduce advancements in high-dimensional approximation for *large-scale* parametric applications, where a single instance of a problem is very computationally expensive due to complex (coupled) model equations or very small, large or complicated geometric extends. More specifically, the applications under consideration are

- Bayesian inference for a problem in medical imaging,
- second moment analysis for an elliptic PDE with random input and
- machine learning for solutions of quantum chemistry calculations.

Within these parametric applications, we have to numerically solve the underlying non-parametric problem (e.g. a PDE). That is, we need a numerical approximation scheme for a large-scale application problem. Usually, the design and implementation of such schemes is a strong challenge by itself. Therefore, one key requirement of all high-dimensional approximation methods in this thesis is to *reuse* existing numerical software. To this end, all methods will either use *solution snapshots* or they will be applied within a given numerical software immediately after discretization.

Due to the high dimensionality of the parameter space, standard approximation techniques (e.g. tensor-product interpolation) that use snapshots will be hit by the *curse of dimensionality*, i.e. an exponential growth of the number of expensive problem instances to be solved for growing parameter dimension d. While solving a *single* large-scale problem is computationally expensive, solving *many* large-scale problems, seems to be barely tractable. Therefore, high-dimensional approximation techniques have to be developed and applied that reuse existing numerical software *and* weaken or break the curse of dimensionality wrt. the parameter space.

The high-dimensional approximation techniques introduced or applied in this thesis include but are not limited to

- Monte-Carlo-type techniques for the solution of the Bayesian inference problem,
- an algebraically sparse grid combination technique for the second moment analysis,
- a sparse grid combination technique for multi-fidelity kernel ridge regression in the quantum chemistry application and
- hierarchical matrices for the approximation of Vandermonde-type matrices from linear systems of equations in kernel-based approximation.

Contents

1 Introduction

1.1 Background

Numerical approximation of high-dimensional problems based on models like partial differential equations (PDEs) is a key tool in research and development. High dimensionality in a given application can be based on an underlying physical problem of high dimensionality (e.g. present in complex plasma physics models) or it can be based on a high-dimensional parameter space appearing in uncertainty quantification, inverse problems, optimization and machine learning. The objective of this thesis is to focus on the latter case and to introduce progress in high-dimensional approximation for *large-scale* parametric applications, where a single instance of a problem is very computationally expensive due to complex (coupled) model equations or very small, large or complicated geometric extends.

For a given parameter space $\Gamma \subset \mathbb{R}^d$ with d being large and $\mathcal{D} \subset \mathbb{R}^D$ a physical space(-time) domain, the objective in parametric problems is to find a solution $\boldsymbol{u} : \Gamma \times \bar{\mathcal{D}} \to \mathbb{R}^r$ such that

$$\mathcal{L}[\boldsymbol{a}]\boldsymbol{u} = \boldsymbol{f}[\boldsymbol{b}] \qquad \text{in } \Gamma \times \bar{\mathcal{D}} \tag{1.1}$$

holds with $\mathcal{L}$ being a possibly non-linear operator. $\boldsymbol{a}, \boldsymbol{b} : \Gamma \times \bar{\mathcal{D}} \to \mathbb{R}^s$ might characterize coefficients in $\mathcal{L}$ and a parametrization of the right-hand side. For simplicity, we skip appropriate boundary and initial conditions for now. In many applications, the objective is further to extract a specific quantity of interest $\boldsymbol{\pi}$ as a function of the solution $\boldsymbol{u}$. Such quantities of interests might be linear functionals or point evaluations. Figure 1.1 gives a generic schematic overview of this general parametric problem, assuming that (1.1) is a PDE.

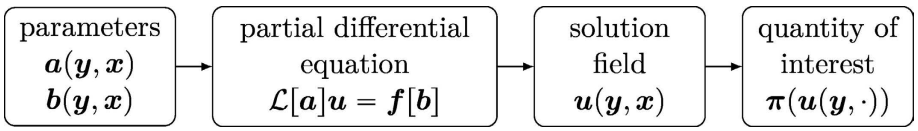


Figure 1.1: Schematic overview of a high-dimensional parametric PDE.

1.1.1 Application scenarios

(Forward) uncertainty quantification. A specific instance of the above problem is *(forward) uncertainty quantification* or *statistical moment estimation*, in which we solve *random* PDEs, i.e. PDEs with random coefficients, random boundary conditions, random domains or random loading, and approximate statistical moments of a random solution for some given random input. That is, after introducing a truncated Karhunen-Loève expansion for the random input variables / fields, cf. [Loè78], we aim at solving (1.1) ρ-almost surely for a given probability

space $(\Gamma, \mathcal{B}, \rho\, d\boldsymbol{y})$ with $\Gamma \subset \mathbb{R}^d$ a parameter space with a dimensionality depending on the number of terms used in the Karhunen-Loève expansion, $\rho : \Gamma \to \mathbb{R}_+$ a probability density function and $\mathcal{B}$ a Borel σ-algebra.

In uncertainty quantification, the special interest is to compute statistical quantities, such as the mean

$$\mathbb{E}\left[\boldsymbol{u}\right](\boldsymbol{x}) := \int_\Gamma \boldsymbol{u}(\boldsymbol{y}, \boldsymbol{x})\rho(\boldsymbol{y})d\boldsymbol{y}$$

or the two-point correlation (e.g. for a scalar u)

$$\mathrm{Cor}\left[u\right](\boldsymbol{x}, \boldsymbol{x}') := \int_\Gamma u(\boldsymbol{x}, \boldsymbol{y})u(\boldsymbol{x}', \boldsymbol{y})\rho(\boldsymbol{y})d\boldsymbol{y} \tag{1.2}$$

of the solution field. Alternatively, these quantities can be computed for a given quantity of interest of the random solution, e.g. $\mathbb{E}\left[\boldsymbol{\pi}(\boldsymbol{u})\right]$. Figure 1.2 summarizes this type of problem.

In this thesis, we consider the computation of (1.2) for an elliptic PDE with random loading.

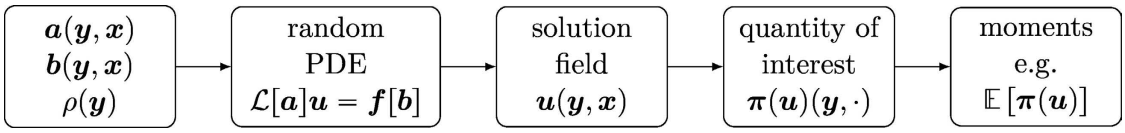


Figure 1.2: Abstract view of a statistical moment analysis.

(Backward) uncertainty quantification. In *(backward) uncertainty quantification* or *(Bayesian) inference* [Stu10, RC15], (the distribution of) the parameter fields $\boldsymbol{a}$ or $\boldsymbol{b}$ are deduced for a given (random) PDE, measurements $\{\pi_1, \pi_2, \ldots, \pi_M\}$ and an observation operator $\mathcal{H}$ that maps the solution of the (random) PDE to observation space, i.e. the space of the measurements. Figure 1.3 visualizes this type of problem. Obviously, the solution of a parametrized PDE as in equation (1.1) is one part of this important application scenario.

In this thesis, we will consider a medical imaging task as Bayesian inference problem.

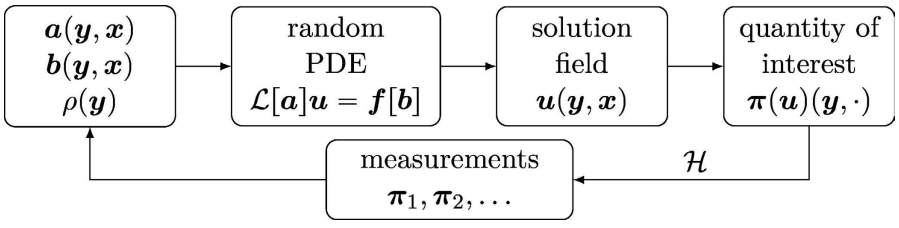


Figure 1.3: Abstract view of a Bayesian inference problem.

Machine learning. The third high-dimensional application discussed in this thesis is *machine learning* for the solutions of computational models / simulations. In this field of growing importance, results of very expensive simulations for large classes of input parameters shall be predicted using a machine learning model that was *trained* with evaluations of a computational

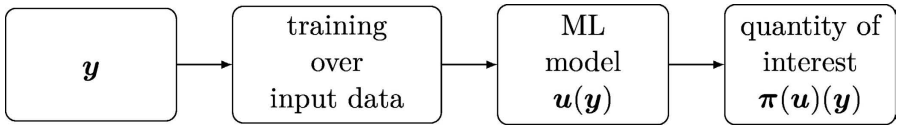


Figure 1.4: Abstract view of a machine learning regression problem.

model for different input parameters. As we will see, cf. Figure 1.4 and Remark 1.5, this task can also be interpreted as a special case of solving a parametric problem.

In this thesis, we will discuss this topic for the important application of quantum chemistry, where we predict chemical properties of unseen molecules based on numerical calculations for other molecules.

1.1.2 Challenges in approximation of high-dimensional large-scale applications

All beforehand mentioned application scenarios require to solve (1.1). In order to find an approximate solution of (1.1), we will certainly have to numerically solve the underlying non-parametric problem. That is, we need a numerical approximation scheme for a large-scale application problem. Usually, the design and implementation of such schemes, is a strong challenge by itself. Therefore, one key requirement of all high-dimensional approximation methods will be to *reuse* existing numerical software. To this end, all methods will either use *solution snapshots* $\boldsymbol{u}(\boldsymbol{y}_i, \cdot)$ for some inputs $\boldsymbol{y}_i$ and derived parameters $\boldsymbol{a}_i := \boldsymbol{a}(\boldsymbol{y}_i, \cdot)$, $\boldsymbol{b}_i := \boldsymbol{b}(\boldsymbol{y}_i, \cdot)$, or they will be applied within a given numerical software immediately after discretization.

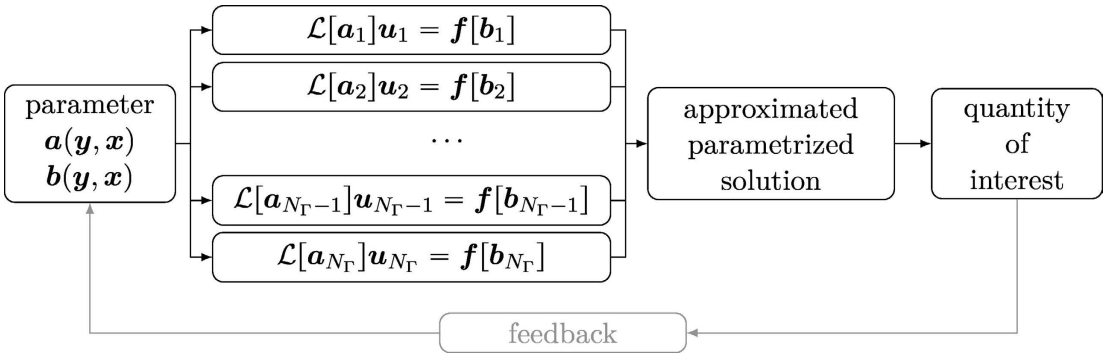


Figure 1.5: Generalized view of the snapshot-based approximation of a parametric problem.

The calculation of approximations based on solution snapshots is visualized in Figure 1.5. Depending on the scenario and the applied methods, (1) a set of problem instances is derived from a usually *high-dimensional* parameter space, (2) the underlying PDE (or other type of problem) is solved for each instance, (3) a parametrized solution (*response surface*) is reconstructed by high-dimensional approximation in the parameter space, (4) a derived quantity is computed and — maybe — (5) the derived quantity is used to start the process from the beginning. While solving a *single* large-scale problem is computationally expensive, solving an application scenario as in Figure 1.5, with many large-scale problems, seems to be barely tractable.

The given situation becomes even more difficult knowing that standard discretizations in the parameter space Γ will result in the *curse of dimensionality*, i.e. an exponential growth of the number of expensive problem instances to be solved for growing parameter dimension d.

Therefore, high-dimensional approximation techniques have to be applied that reuse existing numerical software *and* weaken or break the curse of dimensionality with respect to the parameter space.

1.1.3 Objectives of this thesis

In *high-dimensional approximation*, several strategies are available to reduce the computational work in presence of high dimensionality. The general objective of this thesis is the adaptation, improvement and new development of such strategies for large-scale applications. The specific contributions of this thesis are discussed along the following overview of such strategies.

Function approximation. One approach to approximate (1.1) is given by function approximation for the output quantity of interest. In *sampling* by e.g. Monte Carlo or quasi-Monte Carlo methods, output quantities of interest that are based on an integral can be approximated. Monte Carlo methods are dimension-robust, but have low algebraic convergence rates. More general output quantities of interest can be approximated by *interpolation / collocation* with with e.g. (sparse) tensor-products of univariate polynomials, i.e. sparse grid approximation [BG04], or scattered data approximation / approximation in reproducing kernel Hilbert spaces (RKHS) [Wen04]. Such techniques have, compared to classical Monte Carlo methods, higher requirements on the parametric regularity of the solutions $\boldsymbol{u}$. However, they can achieve much better algebraic or even exponential convergence rates. Other techniques in this field are, e.g., regression or best N-term approximation.

Note that all discussed techniques use solution snapshots. Therefore they are of high interest in context of large-scale applications.

In this thesis, we will apply Monte-Carlo type techniques for the solution of the inference problem in medical imaging. Moreover, function approximation in reproducing kernel Hilbert spaces will be made much cheaper from a computational point of view both in terms of the cost of the individual snapshots by the introduction of the *sparse grid combination technique* (discussed later) to kernel-based approximation and in terms of the actual cost to solve the approximation problem by a change of methodology in the *hierarchical matrix approach* (discussed later).

Hierarchical approximation. In *multi-level* approximation techniques, several levels of discretizations of a given problem are considered. These multiple levels can either be used to introduce space- / time-adaptivity in the discretization, or they can be used in context of *multigrid methods* to introduce linear solvers with optimality properties for discretized problems. A very appealing extension of this approach are so-called *multi-level frame* discretizations [Dah97, HSS08]. This approach, going back to the BPX preconditioner [BPX90], combines basis functions from several levels into one large generating system [Gri94] or frame, giving a way to further generalize the multi-level idea. Space- / time-adaptivity and optimal linear solvers give a means to solve high-dimensional problems at less work and at lower complexity. Moreover, multi-level (frame) discretizations are used in specific high-dimensional multi-level

approximations that will be discussed in the next paragraph. One difficulty of standard *geometric* multi-level (frame) approximations is their application *within* the geometric discretization of a given, e.g., PDE. Therefore, they are not easily applicable to existing numerical software for large-scale applications.

To overcome this, the *algebraic* construction of multi-level frames is introduced in this thesis. It is purely based on the linear system constructed *after* the discretization of e.g. an elliptic PDE. Hence, it can be easily applied to existing numerical software and is, among other properties, independent of potentially complex geometries. The new algebraic construction is based on techniques known from *algebraic multigrid* [BMR82]. One application of this new approach will be optimal-complexity linear solvers. Also, based on this work, a new algebraic high-dimensional approximation technique is constructed.

Sparse tensor-product approximation. This type of techniques combines ideas from high-dimensional function approximation with hierarchical approximation techniques. In the *multi-level (quasi-)Monte Carlo approach* [Hei01, Gil15], strong work reductions are achieved by efficiently combining solution samples on different discretization levels. A related technique is the *sparse grid combination technique* [GSZ92] that allows to build sparse grid approximations using solutions discretized on anisotropic regular grids. The idea of the sparse grid combination technique can be further generalized to a multi-level / multi-fidelity approximation technique of problems with multi-level hierarchies in different spaces, e.g. parametric spaces, time space, physical domain space [RG18]. Here, it provides a combination methodology that achieves an optimal balancing of costs of combined solutions $\boldsymbol{u}^{(\boldsymbol{\ell})}$ being discretized on different levels per underlying space. From a multi-level Monte Carlo perspective, this general idea is often known as *multi-index approximation* [HANTT16]. The sparse grid combination technique and multi-index approximations use multi-level sampling or multi-level collocation-type function approximation in parametric space, while requiring multi-level discretizations in e.g. physical space.

In this thesis, *algebraic multi-level frames* are used to construct an *algebraic sparse grid combination technique*. It allows to build a sparse grid combination technique approximation by only using knowledge from a linear system that is assembled from a given numerical software. This approach is specifically applied for the approximate solution of large-scale complex-geometry elliptic tensor product problems. Such problems show up in the second moment analysis, i.e. the covariance estimation, for elliptic random PDEs. Due to the new algebraic approach, this analysis becomes feasible for existing numerical software and for problems with complex geometries.

An additional contribution of this thesis is the introduction of the generalized sparse grid combination / multi-index approach to kernel-based approximation or *kernel ridge regression* [VSL+15]. That is, this approach is introduced to the field of regression in machine learning. The specific machine learning application is the prediction of atomization energies for molecules based on trained quantum chemistry calculations. Within this approach, multi-level hierarchies are introduced in the learning part (i.e. the number of molecules) and in the quantum chemistry part (i.e. the quantum chemistry calculation model and the *basis set size* used in the calculations). For a fixed target prediction error, a strong reduction in required expensive training samples is achieved.

Low-rank approximation for discrete high-dimensional data. In this field, one considers the approximation of discrete data that is either stored in matrices or in higher-order tensors. High dimensionality is present either due to a functional dependence of the discrete data on a higher-dimensional input or by the high dimensionality / order of the tensor. A typical example for matrices with entries depending on high-dimensional inputs are interpolation matrices from high-dimensional scattered data interpolation or approximation. Higher-order tensors can be used to describe solutions to parametric problems in a discrete sense. In all the beforehand mentioned situations, efficient data structures become very important. Tensors have to be represented in appropriate *tensor formats* [Hac12], while scattered data is structured in e.g. *kD-trees*, *quad trees* or by *space filling curves*. Given these well-structured data representations, low-rank approximation techniques such as *adaptive cross approximation (ACA)* [Beb00], *singular value decomposition (SVD)* or the *pivoted Cholesky factorization* [HPS12] can be used to efficiently reduce the required computational cost for computations on the discrete data, while keeping a prescribed fixed approximation tolerance.

In this thesis, advancements on the efficient approximation of e.g. interpolation matrices from interpolation in reproducing kernel Hilbert spaces will be made. More specifically, space filling curves are introduced to the ACA-based *hierarchical ($\mathcal{H}$) matrix approach* [Hac15] that allows to efficiently approximate matrices with entries corresponding to evaluations of *kernel* functions, which are smooth away from the diagonal. This change in the underlying data structure will result in a restructuring of the hierarchical matrix algorithm that allows this method to be parallelized much better. Thereby, kernel-based function approximation will become pre-asymptotically much faster.

1.1.4 Overview

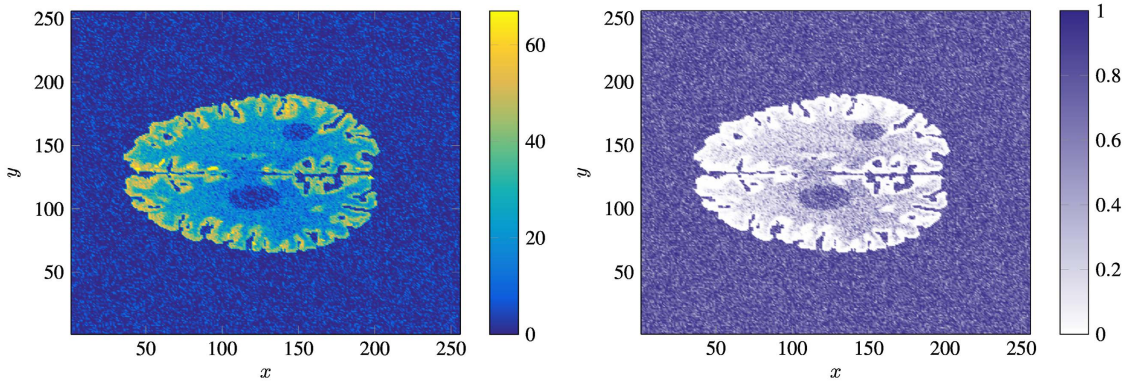
The remainder of this introduction section is structured as follows. In Section 1.2, we start by briefly introducing the three major large-scale applications present in this thesis. Afterwards, in Sections 1.3 to 1.7, the discussed high-dimensional approximation techniques will be concisely introduced from a mathematical point of view. Hints to the new contributions of this thesis will be pointed out. Finally, Section 1.8 gives a structured overview (with statement on the author's own contribution for non single-authored works) over each article and preprint that is the basis of this cummulative habilitation.

1.2 High-dimensional problems in practice

1.2.1 Perfusion estimation in dynamic contrast-enhanced imaging

The first application in this thesis is in the field of (backward) uncertainty quantification, that is, inference in medical imaging. The aim is to infer a time-stationary perfusion (blood flow) information $\boldsymbol{p} \in \mathbb{R}^{N_{voxel}}$, cf. Figure 1.6(a), given the (assumed to be exactly known) inflow concentration c_{art} of a contrast agent and an "observation matrix" $\boldsymbol{C}$ of time-dependent volumetric images of a patient's tissue. Formally, the blood perfusion in a given voxel j can be evaluated as quantity of interest of a so-called response function k_j as

$$p_j := p(k_j) := {\rho_j}^{-1} k_j(0)\,.$$



(a) approximated perfusion $\overline{\boldsymbol{p}}$ (b) approx. probability of low perfusion ($\overline{\boldsymbol{p}} < 10$)

Figure 1.6: In the inference application, perfusion information (i.e. blood flow rates) are estimated from noisy medical imaging data in a Bayesian framework.

The underlying deterministic inference problem reads as follows: For given total measurement time $T \in \mathbb{R}$, arterial inflow $c_{art} : [0,T] \to \mathbb{R}_{\geqslant 0}$, measurement / observation times $t_1^{obs} < t_2^{obs} < \ldots < t_{N_{obs}}^{obs}$ and observation matrix $\boldsymbol{C}$, composed from vectors $\boldsymbol{c}_i \in \mathbb{R}^{N_{voxel}}$, $i \in \{1,\ldots,N_{obs}\}$, we aim at computing a vector $\boldsymbol{k} = (k_1,\ldots,k_{N_{voxel}})^T$ of response functions $k_j : [0,T] \to \mathbb{R}$ and the derived quantity of interest $\boldsymbol{p} = (p_1,\ldots,p_{N_{voxel}})^\top$ with $p_j = k_j(0)/\rho_j$ such that

$$c_{j,i} \approx \int_0^T c_{art}(\tau) k_j(t_i^{obs} - \tau) d\tau + e_{j,i}(c_{art}, k)\,, \qquad j \in \{1,\ldots,N_{voxel}\},\ i \in \{1,\ldots,N_{obs}\}\,. \quad (1.3)$$

This problem is under-determined with the given requirements. Furthermore, we have not specified the nature of the error term $e_{j,i}$. This is why we used the notion "$\approx$".

The main contribution of this thesis to this problem is to reformulate it as a Bayesian sequential data assimilation task, which is solved on synthetic data by sampling-based methods, i.e. the ensemble-based Kalman filter. The Bayesian view allows to compute additional probabilistic reliability information, cf. Figure 1.6(b), which might be crucial for diagnosis, in the long run.

1.2.2 Second moment analysis for elliptic problems

The approximation of the mean of a partial differential equation has been studied for many problems. However, the approximation of the statistical quantity *two-point correlation*, cf. (1.2), is usually avoided, since it involves to approximate a bivariate function in the full physical space $\mathcal{D}$. That is, a problem in $2D$ dimensions has to be solved. Even for small dimensions $D = 2, 3$, this can become very computationally expensive.

Let us exemplify this for an elliptic problem with random loading. We consider a given deterministic matrix-valued coefficient function $\boldsymbol{A} : \mathcal{D} \to \mathbb{R}^{D\times D}$ and aim at finding $u : \Gamma \times \bar{\mathcal{D}} \to$

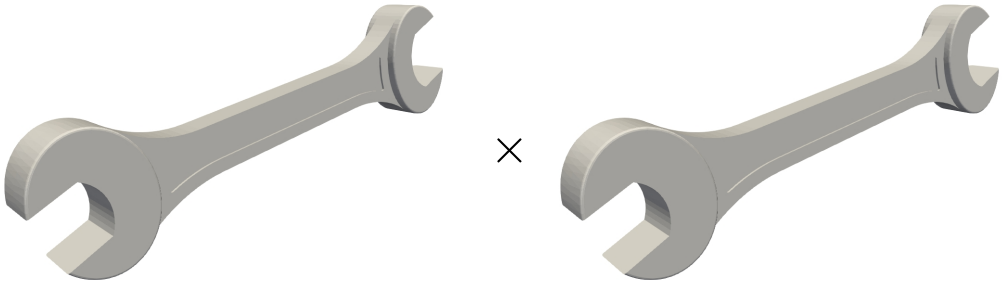


Figure 1.7: In the second moment analysis example, we consider the sparse black-box approximation of the two-point correlation, e.g. on spanner geometries.

$\mathbb{R}$ such that it holds almost surely for all $\boldsymbol{y} \in \Gamma$

$$\begin{aligned} -\mathrm{div}_{\boldsymbol{x}}(\boldsymbol{A}\nabla_{\boldsymbol{x}} u(\boldsymbol{y})) &= f(\boldsymbol{y}) && \text{in } \mathcal{D}\,, \\ u(\boldsymbol{y}) &= 0 && \text{on } \partial\mathcal{D}\,. \end{aligned}$$

Under appropriate assumptions, it is known (e.g. from [ST03]) that the two-point correlation of u considered as function $\mathrm{Cor}\,[u] : \bar{\mathcal{D}} \times \bar{\mathcal{D}} \to \mathbb{R}$ can be computed by solving the *deterministic* boundary value problem

$$\begin{aligned} (\mathrm{div}_{\boldsymbol{x}} \otimes \mathrm{div}_{\boldsymbol{x}'})\,((\boldsymbol{A}_{\boldsymbol{x}} \otimes \boldsymbol{A}_{\boldsymbol{x}'})\,(\nabla_{\boldsymbol{x}} \otimes \nabla_{\boldsymbol{x}'})\,\mathrm{Cor}\,[u]) &= \mathrm{Cor}\,[f] && \text{in } \mathcal{D} \times \mathcal{D}\,, \\ \mathrm{Cor}\,[u] &= 0 && \text{on } \partial(\mathcal{D} \times \mathcal{D})\,. \end{aligned}$$

This is an elliptic boundary value problem on the $2D$-dimensional domain $\bar{\mathcal{D}} \times \bar{\mathcal{D}}$. If we introduce a finite-dimensional approximation for real-valued functions on $\mathcal{D}$ with a basis of size $N_{\mathcal{D}}$, representing the solution $\mathrm{Cor}\,[u]$ will already require $N_{\mathcal{D}}^2$ degrees of freedom. Then, even with multigrid, we need $O(N_{\mathcal{D}}^2)$ operations to get an approximate solution. However, following the lines of [ST03, HPS13], it is possible to introduce a sparse approximation for $\mathrm{Cor}\,[u]$ leading to an approximation method with almost linear complexity in $N_{\mathcal{D}}$, up to polylogarithms. Nevertheless, the sparse construction proposed in [ST03, HPS13] usually requires to have a geometrical multi-level discretization of $N_{\mathcal{D}}$ at hand. This makes it almost impossible to apply this approach in a black-box fashion.

For the sake of simplicity, we restrict ourselves to the case of $\boldsymbol{A} := \boldsymbol{I}$, i.e. the identity matrix, and aim at solving the model problem

$$\begin{aligned} (\mathrm{div}_{\boldsymbol{x}} \otimes \mathrm{div}_{\boldsymbol{x}'})\,((\nabla_{\boldsymbol{x}} \otimes \nabla_{\boldsymbol{x}'})\,U) &= F && \text{in } \mathcal{D} \times \mathcal{D}\,, \\ U &= 0 && \text{on } \partial(\mathcal{D} \times \mathcal{D})\,, \end{aligned}$$

with the load function $F : \bar{\mathcal{D}} \times \bar{\mathcal{D}} \to \mathbb{R}$ and the solution $U : \bar{\mathcal{D}} \times \bar{\mathcal{D}} \to \mathbb{R}$. For this problem, we discuss a new *algebraic* multi-level construction that shows the same preferable asymptotic computational complexity as the sparse approaches in [ST03, HPS13]. However, this approach only relies on information given by the stiffness matrix and the mass matrix given by a univariate elliptic boundary value problem on $\mathcal{D}$, in order to construct a multi-level hierarchy for

$\mathcal{D} \times \mathcal{D}$. This allows to introduce this sparse approach to a much broader class of problems, such as problems on complex geometries, cf. Figure 1.7, in a black-box fashion.

1.2.3 Atomization energy estimation in quantum chemistry

In computational quantum chemistry, which we discuss following the lines of [Ham09], or more specifically in electronic structure calculations, one is interested in the approximate solution of the stationary Schrödinger equation

$$H_{\boldsymbol{\mathcal{M}}}\Psi = E\Psi\,. \tag{1.4}$$

This equation is an eigenvalue problem in which (E, Ψ) is a pair of eigenvalue E and eigenfunction Ψ for the *Hamilton operator* $H_{\boldsymbol{\mathcal{M}}}$. The Hamilton operator, roughly speaking, encodes the properties of a molecule $\boldsymbol{\mathcal{M}}$, where we use atomic units for all remaining quantities. A molecule $\boldsymbol{\mathcal{M}}$ consists of N_{el} electrons $\boldsymbol{\mathcal{E}}_p$ and N_{nuc} nuclei $\boldsymbol{\mathcal{N}}_q$. The pth electron is characterized by its position $\boldsymbol{x}_p \in \mathbb{R}^3$ while the qth nucleus is characterized by the triple $(\boldsymbol{r}_q, m_q, z_q) \in \mathbb{R}^3 \times \mathbb{R} \times \mathbb{R}$ with position, mass and atomic number. The Hamilton operator of the stationary Schrödinger equation is given by

$$\begin{aligned} H_{\boldsymbol{\mathcal{M}}} := &-\frac{1}{2}\sum_{p=1}^{N_{el}} \Delta_{\boldsymbol{x}_p} - \sum_{p=1}^{N_{el}}\sum_{q=1}^{N_{nuc}} z_q \frac{1}{\|\boldsymbol{x}_p - \boldsymbol{r}_q\|_2} + \sum_{p=1}^{N_{el}}\sum_{p'>p}^{N_{el}} \frac{1}{\|\boldsymbol{x}_p - \boldsymbol{x}_{p'}\|_2} \\ &+ \sum_{q=1}^{N_{nuc}}\sum_{q'>q}^{N_{nuc}} z_q z_{q'} \frac{1}{\|\boldsymbol{r}_q - \boldsymbol{r}_{q'}\|_2} - \frac{1}{2}\sum_{q=1}^{N_{nuc}} \frac{1}{2m_q}\Delta_{\boldsymbol{r}_q}\,. \end{aligned} \tag{1.5}$$

Here, $\Delta_{\boldsymbol{x}_p}$ and $\Delta_{\boldsymbol{r}_p}$ are the Laplace operators with respect to the named coordinate directions. The terms of the operator describe the kinetic energy of the electrons, the potential energy of the interactions between electrons and nuclei, the potential energy of the repulsion between the electrons, the potential energy of the repulsion between the nuclei and the kinetic energy of the nuclei (in this order). The smallest eigenvalue $E_{\min}$ of $H_{\boldsymbol{\mathcal{M}}}$ is the *ground state energy* of molecule $\boldsymbol{\mathcal{M}}$, which we denote as $E_{\min}(\boldsymbol{\mathcal{M}})$.

We aim at approximating the parametric Schrödinger equation (1.4) in which the parameter space Γ is a subset of all possible molecules and where the quantity of interest π is the so-called *atomization energy*, cf. Figure 1.8, of a given molecule $\boldsymbol{\mathcal{M}}$. The atomization energy of molecule $\boldsymbol{\mathcal{M}}$ describes the energy that is necessary to cut all atom bonds in the molecule. This means, it is, up to change of sign, the difference between the ground state energy $E_{\min}(\boldsymbol{\mathcal{M}})$ for molecule $\boldsymbol{\mathcal{M}}$ and the sum $\sum_{q=1}^{N_{nuc}} E_{\min}(\boldsymbol{\mathcal{N}}_q)$ over the ground state energies of each individual nucleus.

Practical approaches to approximately compute the ground state energy require further simplifications of the Hamilton operator $H_{\boldsymbol{\mathcal{M}}}$. One usually solves the simplified *electronic Schrödinger equation* based on the *Born-Oppenheimer approximation*. Then, there are many different methods, such as *Hartree-Fock*, *Møller-Plesset 2* and *coupled cluster*, which provide approximate solutions to the ground state energy. As usually, better approximations lead to a much higher computational cost. Moreover, each method needs a *basis set*. Roughly speaking, a larger basis set size leads to a better approximation with an upper bound of the achievable accuracy imposed by the method in the *basis set size limit*. The ultimate goal is to approximate

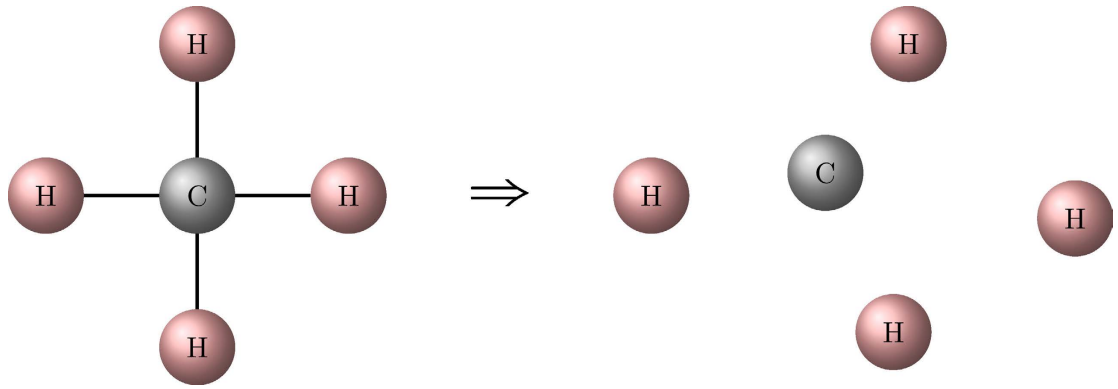


Figure 1.8: In the machine learning application, we approximate *atomization energies* of molecules, i.e. the energy necessary to break a molecule into its (unbound) atoms.

the atomization energy of a molecule with *chemical accuracy*, that is, at an accuracy which corresponds to the best possible measurement accuracy in chemical experiments. However, such calculations become computationally very expensive.

The main contribution of this thesis, in the above context, is the approximation of the mapping between molecule and atomization energy by a kernel ridge regression model at chemical accuracy. To overcome the exceptional computational requirement for this application, a new multi-fidelity kernel ridge regression approach is introduced that is based on the sparse grid combination technique. It strongly reduces the amount of very expensive training sample calculations by adding many cheap training samples.

1.3 Function approximation

We observe that we can rewrite our general problem (1.1) such that the solution $\boldsymbol{u}$ is a Banach space-valued function

$$\boldsymbol{u} : \Gamma \to \mathcal{V}$$

with elements $\boldsymbol{v} \in \mathcal{V}$ of the form $\boldsymbol{v} : \bar{\mathcal{D}} \to \mathbb{R}^r$. That is, an approximate solution of (1.1) can be interpreted as the approximation of a Banach space valued (high-dimensional) function.

Recall, that we are often interested to compute real-valued quantities of interest $\pi(\boldsymbol{u})$, i.e. real-valued numbers that are even no longer dependent on the physical space. Therefore, we will restrict ourselves to the discussion of high-dimensional function approximation for real-valued functions $u : \Gamma \to \mathbb{R}$.

In the following, we discuss a series of different approximation approaches to find for functions $u : \Gamma \to \mathbb{R}$, which are elements of some function space $\mathcal{F}$ to be specified later, approximations $\mathcal{A}_V u$ in some finite-dimensional subspace $V \subset \mathcal{F}$. We call $\mathcal{A}_V u$ an *approximation* for u, if it holds

$$\mathcal{A}_V u \approx u \,.$$

Note that the exact definition of *approximation* will depend on the type of approximation that we will use. We will replace the general operator $\mathcal{A}_V$ by specific versions, e.g. $\mathcal{B}_V$ for best approximation, $\mathcal{I}_V$ for interpolation, etc.

Remark 1.1 (Sampling-based techniques)**.** *In this overview, we stick to the discussion of high-dimensional* function approximation *for general output quantities of interest. Sampling-based Monte-Carlo type methods, which can only be applied in context of integral-type quantities of interest, are, e.g., discussed in [Caf98].*

Relation to achieved results

A Monte-Carlo type method is used in Chapter 2 in order to approximate a *Kalman filter* for Bayesian inference in a medical imaging application.

1.3.1 Best approximation

Let $\mathcal{F}$ be a Hilbert space of functions of type $u : \Gamma \to \mathbb{R}$ with scalar product $\langle \cdot, \cdot \rangle$ and the induced canonical norm $\| \cdot \|$. We consider $V \subset \mathcal{F}$ to be a finite-dimensional linear subspace of $\mathcal{F}$. In (linear) *best approximation* for a function $u \in \mathcal{F}$, we aim at finding a function $\mathcal{B}_V u \in V$ such that it holds

$$\|u - \mathcal{B}_V u\| = \inf_{v \in V} \|u - v\| .$$

The existence and uniqueness of such an infimum is guaranteed, since V is a finite-dimensional linear subspace of $\mathcal{F}$ [HH94]. We further know [Ran17] that the above statement on the best approximation is equivalent to

$$\langle u - \mathcal{B}_V u, \phi \rangle = 0 \quad \forall \phi \in V . \tag{1.6}$$

Since V is finite-dimensional, we can now choose a (finite) basis, i.e. $V := \operatorname{span}\{\phi_1, \dots, \phi_N\}$, and write

$$(\mathcal{B}_V u)(\boldsymbol{y}) := \sum_{j=1}^{N} \alpha_j \phi_j(\boldsymbol{y}) .$$

With (1.6), we conclude

$$\langle u - \mathcal{B}_V u, \phi \rangle = \langle u - \sum_{j=1}^{N} \alpha_j \phi_j(\boldsymbol{y}), \phi \rangle = \langle u, \phi \rangle - \sum_{j=1}^{N} \alpha_j \langle \phi_j(\boldsymbol{y}), \phi \rangle = 0 \quad \forall \phi \in V .$$

By inserting $\phi := \phi_j$ for $j \in \{1, \dots, N\}$, we end up with the system of linear equations

$$\sum_{j=1}^{N} \langle \phi_j, \phi_i \rangle \alpha_j = \langle u, \phi_i \rangle \quad \forall i \in \{1, \dots, N\} ,$$

which allows to compute the coefficients α_j and hence $\mathcal{B}_V u$.

Example 1.1 (L^2 projection)**.** *For $\mathcal{F} = L^2(\Gamma)$, i.e. the Lebesgue space of all square integrable functions $u : \Gamma \to \mathbb{R}$, equipped with the standard L^2 norm and scalar product, the best approximation $\mathcal{B}_V u$ is the L^2* projection *$\mathcal{P}_V u$ of u onto V.* △

1.3.2 Interpolation

In *interpolation*, we again consider to approximate a function $u \in \mathcal{F}$ by a function from an N-dimensional subspace V. However, we additionally require u to be point-evaluable. Moreover, we introduce N abscissas

$$X := \{\boldsymbol{y}_1, \dots, \boldsymbol{y}_N\}\,.$$

Then, we introduce the *interpolant* $\mathcal{I}_{V,X}u$ as the function that fulfills for each abscissa $\boldsymbol{y}_i$ the interpolation condition

$$(\mathcal{I}_{V,X}u)(\boldsymbol{y}_i) = u(\boldsymbol{y}_i)\,.$$

Once again, we use a finite basis $\{\phi_1, \dots, \phi_N\}$ for V and make the ansatz $(\mathcal{I}_{V,X}u)(\boldsymbol{y}) := \sum_{j=1}^N \alpha_j \phi_j(\boldsymbol{y})$ to end up with a system of linear equations of the form

$$\sum_{j=1}^N \alpha_j \phi_j(\boldsymbol{y}_i) = u(\boldsymbol{y}_i) \quad \forall i \in \{1, \dots, N\}\,,$$

which allows to compute the coefficients α_j and thereby $\mathcal{I}_{V,X}u$.

An alternative way to formulate interpolation is given by the use of a *Lagrange basis* $\{L_i\}_{i=1}^N$ for V such that

$$L_i(\boldsymbol{y}_j) = \begin{cases} 1, & i = j\,, \\ 0, & i \neq j\,. \end{cases}$$

For given data $\{(\boldsymbol{y}_i, u(\boldsymbol{y}_i))\}_{i=1}^N$, the interpolant becomes

$$(\mathcal{I}_{V,X}u)(\boldsymbol{y}) = \sum_{i=1}^N u(\boldsymbol{y}_i) L_i(\boldsymbol{y})\,.$$

To determine the Lagrange basis, we express each L_i in terms of the finite-dimensional basis according to $L_i(\boldsymbol{y}) := \sum_{j=1}^N \alpha_j^{(i)} \phi_j(\boldsymbol{y})$. Then, we can compute the $\alpha_j^{(i)}$ by solving

$$\sum_{j=1}^N \alpha_j^{(i)} \phi_j(\boldsymbol{y}_i) = \delta_{i,j}\,.$$

Remark 1.2 (Collocation for parametric problems)**.** *We can easily use function interpolation by a Lagrange basis to solve parametric problems. In the context of parametric problems, this approach is called* collocation *[BNT10].*

Let us briefly recall (1.1)*, where we aim at solving*

$$\mathcal{L}\left[\boldsymbol{a}(\boldsymbol{y}, \boldsymbol{x})\right] \boldsymbol{u}(\boldsymbol{y}, \boldsymbol{x}) = \boldsymbol{f}\left[\boldsymbol{b}(\boldsymbol{y}, \boldsymbol{x})\right](\boldsymbol{y}, \boldsymbol{x}) \quad \text{in } \Gamma \times \bar{\mathcal{D}}\,.$$

We then generate data $\{(\boldsymbol{y}_i, \boldsymbol{u}(\boldsymbol{y}_i, \cdot))\}$ *by solving* N *non-parametric problems*

$$\mathcal{L}[a(\boldsymbol{y}_i, \boldsymbol{x})]\boldsymbol{u}(\boldsymbol{y}_i, \boldsymbol{x}) = \boldsymbol{f}[\boldsymbol{b}(\boldsymbol{y}_i, \boldsymbol{x})](\boldsymbol{y}_i, \boldsymbol{x}) \quad \text{in } \bar{\mathcal{D}} \quad \forall i \in \{1, \dots, N\}\,.$$

The approximate solution $\mathcal{I}_{V,X}\boldsymbol{u}$ of the parametric problem finally becomes

$$\left(\mathcal{I}_{V,X}\boldsymbol{u}\right)(\boldsymbol{y},\boldsymbol{x}) := \sum_{i=1}^{N} \boldsymbol{u}(\boldsymbol{y}_i,\boldsymbol{x}) L_i(\boldsymbol{y})\,.$$

1.3.3 Regression

In interpolation, we considered a situation, in which we have the same amount of data $\{(\boldsymbol{y}_i, u(\boldsymbol{y}_i))\}_{i=1}^N$ as the dimensionality of the subspace V. We now consider the situation, in which we have M data points with $M > N$. In this case, we have to solve a *regression* problem. We here briefly discuss *least squares approximation* [HH94]. That is, we seek a function $\mathcal{R}_V u := \sum_{j=1}^N \alpha_j \phi_j(\boldsymbol{y})$ with a coefficient vector $\boldsymbol{\alpha}$ such that it holds

$$\boldsymbol{\alpha} := \operatorname*{argmin}_{\boldsymbol{\alpha}'} \sum_{i=1}^{M} \left| u(\boldsymbol{y}_i) - \sum_{j=1}^{N} \alpha'_j \phi_j(\boldsymbol{y}_i) \right|^2 .$$

By setting up the matrix $\boldsymbol{A} \in \mathbb{R}^{M\times N}$ with entries $a_{i,j} := \phi_j(\boldsymbol{y}_i)$ and the vector $\boldsymbol{u}$ with entries $u_i := u(\boldsymbol{y}_i)$, we can compute the coefficients α_j by solving the known to be unstable normal equations

$$\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{\alpha} = \boldsymbol{A}^\top \boldsymbol{u}\,.$$

1.4 Suitable approximation spaces

In the following, we give typical examples of approximation spaces V. We start with two approximation spaces for univariate functions. These give examples for the generalization to multivariate approximation spaces in terms of tensor-products of univariate approximation spaces. Finally two multivariate approximation spaces are provided. We always move from approximation spaces, where we can increase a number of points on a *grid*, i.e. *h-refinement*, to approximation space, where we can increase the polynomial degree, i.e. *p-refinement*.

1.4.1 Univariate piecewise linear polynomials

We restrict ourselves to approximation spaces for functions on closed subsets of $\Gamma \subset \mathbb{R}$. For simplicity, we choose $\Gamma = [-1, 1]$. Moreover, we need a set of points X_ℓ that define the intervals on which the polynomials are given. We define these with respect to a *level* ℓ. Typical choices for X_ℓ with $X_\ell := \{y_i\}_{i=1}^{N_\ell}$ are *uniformly distributed points* with

$$y_i = -1 + (i-1)\cdot h_\ell, \quad h_\ell = \frac{2}{N_\ell - 1}, \quad N_\ell = 2^{\ell-1} + 1, \quad \ell \geqslant 2 \tag{1.7}$$

or *Clenshaw-Curtis points* with

$$y_i = -\cos\frac{\pi(i-1)}{N_\ell - 1}, \quad N_\ell = 2^{\ell-1} + 1, \quad \ell \geqslant 2\,. \tag{1.8}$$

Then, the piecewise linear polynomial approximation space PW_{N_ℓ} becomes

$$V = PW_{N_\ell} := \left\{ u : \Gamma \to \mathbb{R} \middle| u \in C[-1,1],\, u|_{[y_i,y_{i+1}]} \text{ is linear}, i \in \{1,\dots,N_\ell - 1\} \right\}. \tag{1.9}$$

Remark 1.3. *We could easily extend the above approximation space to piecewise polynomials of higher degree or even to splines. However, since these spaces are not used throughout this thesis, we skip their discussion.*

1.4.2 Univariate polynomials

Another important univariate approximation space are *univariate polynomials*. For simplicity, we again choose $\Gamma = [-1,1]$. Then, the finite-dimensional subspace $V := P_N$ of polynomials

$$V = P_N := \left\{ u : \Gamma \to \mathbb{R} \middle| u(y) = \sum_{j=0}^{N} a_j y^j \right\}.$$

is often used for function approximation of smooth functions.

Example 1.2 (Interpolation by univariate polynomials)**.** *For smooth functions $u : [-1,1] \to \mathbb{R}$, we use $V := P_N$ and introduce the set of abscissas*

$$X := \{y_0, y_1, \dots, y_N\} \subset [-1,1].$$

The interpolant $\mathcal{I}_{P_N,X}$ with respect to the monomial basis $\{1, y, y^2, \dots, y^N\}$ and coefficients $\{\alpha_i\}_{i=0}^N$ can be computed by solving the system of linear equations

$$\boldsymbol{A}_{P_N,X}\boldsymbol{\alpha} = \boldsymbol{u},$$

where $u_i := u(y_i)$ and

$$\boldsymbol{A}_{P_N,X} := \begin{pmatrix} 1 & y_0 & y_0^2 & \cdots & y_0^N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & y_N & y_N^2 & \cdots & y_N^N \end{pmatrix}.$$

Note, however, that one would usually avoid the explicit solution of this system of linear equations for computational cost reasons. △

1.4.3 Tensor products of univariate approximation spaces

We use a tensor product construction to extend the above univariate approximation spaces to multivariate approximation spaces. To consider approximation of functions in a full d-dimensional space $\Gamma := \Gamma^{(1)} \times \cdots \times \Gamma^{(d)}$, we take a sequence

$$V^{(1)}, V^{(2)}, \dots, V^{(d)}$$

of d univariate finite-dimensional approximation spaces with

$$\dim V^{(1)} = \ldots = \dim V^{(d)} = N.$$

The resulting tensor product approximation space T_N becomes

$$V = T_N := V^{(1)} \otimes V^{(2)} \otimes \cdots \otimes V^{(d)} .$$

If we have for each space $V^{(i)}$ a basis $\left\{\phi_1^{(i)}, \ldots, \phi_N^{(i)}\right\}$, T_N is given by

$$T_N := \operatorname{span}\left\{ \prod_{i=1}^{d} \phi_{j_i}^{(i)} \middle| (j_1, \ldots, j_d) \in \{1, \ldots, N\}^d \right\} .$$

The general tensor product approximation operator $\mathcal{A}_{T_N}$ is then formally given by

$$\mathcal{A}_{T_N} := \mathcal{A}_{V^{(1)}} \otimes \cdots \otimes \mathcal{A}_{V^{(d)}} .$$

Example 1.3 (Interpolation by tensor products of piecewise linear functions)**.** *A d-variate Lagrange interpolation formula on $\Gamma = [-1, 1]^d$ can easily be derived from the tensor-product of d univariate interpolation formulas*

$$\begin{aligned}(\mathcal{I}_{T_{N_\ell, X}} u)(\boldsymbol{y}) &:= \left(\mathcal{I}^{(1)}_{PW_{N_\ell}, X_\ell^{(1)}} \otimes \mathcal{I}^{(2)}_{PW_{N_\ell}, X_\ell^{(2)}} \otimes \cdots \otimes \mathcal{I}^{(d)}_{PW_{N_\ell}, X_\ell^{(d)}} \right)(\boldsymbol{y}) \\ &= \sum_{j_1=1}^{N_\ell} \cdots \sum_{j_d=1}^{N_\ell} u(\boldsymbol{y_j}) \left(L_{j_1}^{(1)} \otimes \cdots \otimes L_{j_d}^{(d)} \right)(\boldsymbol{y}) ,\end{aligned}$$

with

$$\left(L_{j_1}^{(1)} \otimes \cdots \otimes L_{j_d}^{(d)} \right)(\boldsymbol{y}) := L_{j_1}^{(1)} \left(y^{(1)} \right) \cdot \ldots \cdot L_{j_d}^{(d)} \left(y^{(d)} \right) .$$

The interpolation abscissas are

$$\boldsymbol{y_j} = \left(y_{j_1}^{(1)}, \ldots, y_{j_d}^{(d)} \right) \in X := X_\ell^{(1)} \times \cdots \times X_\ell^{(d)} ,$$

where $\boldsymbol{j} := (j_1, \ldots, j_d)$ is a d-dimensional multi-index. Each individual $y_{j_i}^{(i)}$ is, e.g., given in (1.7)*. The tensor product piecewise linear interpolation formally constructs interpolants from functions evaluations on points $\left(y_{j_1}^{(1)}, \ldots, y_{j_d}^{(d)} \right)$ that are given on the grid X. It is obvious that the number of required function evaluations, for fixed univariate resolution N_ℓ, scales* exponentially *in the number of dimensions. This is the* curse of dimensionality*.* △

1.4.4 Piecewise linear polynomials on triangulations

Tensor product approximations are tied to an underlying space Γ that has tensor product structure. To handle more general parametric spaces Γ, we can approximate functions $\mathcal{F} \subset L^2(\Gamma)$ by piece-wise polynomials on triangulations. That is, we assume to have for Γ a triangulation

$$\mathcal{T} := \{\tau_1, \ldots, \tau_M\} .$$

We can then introduce the finite dimensional subspace

$$V = \mathcal{P}_{\mathcal{T}} := \left\{ u \in C(\Gamma) \middle| \forall \tau \in \mathcal{T}_\ell : u|_\tau \text{ is linear} \right\},$$

i.e. piecewise linear polynomials on the triangulation.

Remark 1.4. *This finite dimensional approximation space is the standard ansatz space for finite element approximations. It is less often used for approximation in the parametric domain* Γ*. However approximation by piecewise polynomials is sometimes used in uncertainty quantification under the notion of* simplex stochastic collocation *[WLB09].*

Relation to achieved results

In Chapters 3 and 4, we discuss the solution of elliptic problems that are discretized by piecewise linear polynomials on triangulations.

1.4.5 Reproducing kernel Hilbert spaces

In tensor product approximation, we are fixed to domains Γ of tensor product structure, while piecewise polynomials on triangulations allow to do function approximation on more general domains. However, still, approximations are connected to a spatial structure, i.e. a mesh. In contrast, *scattered data approximation* can achieve a *meshless* finite-dimensional approximation of functions from a *reproducing kernel Hilbert space*. The underlying theory is discussed following [Wen04].

A *reproducing kernel* k for a general Hilbert space $\mathcal{F}$ is a function $k : \Gamma \times \Gamma \to \mathbb{R}$ such that

1. $k(\cdot, \boldsymbol{y}) \in \mathcal{F}$ for all $\boldsymbol{y} \in \Gamma$,
2. $f(\boldsymbol{y}) = (u, k(\cdot, \boldsymbol{y}))_{\mathcal{F}}$ for all $u \in \mathcal{F}$ and all $\boldsymbol{y} \in \Gamma$.

$\mathcal{F}$ is called *reproducing kernel Hilbert space (RKHS)*, if it has a reproducing kernel $k : \Gamma \times \Gamma \to \mathbb{R}$. A continuous kernel $k : \Gamma \times \Gamma \to \mathbb{R}$ is further *positive semi-definite* on $\Gamma \subseteq \mathbb{R}^d$, if for all $N \in \mathbb{N}$, all pairwise distinct $X = \{\boldsymbol{y}_1, \dots, \boldsymbol{y}_N\} \subseteq \Gamma$, and all $\alpha \in \mathbb{R}^N \backslash \{0\}$, we have $\sum_{j=1}^N \sum_{j'=1}^N \alpha_j \alpha_{j'} k(\boldsymbol{y}_j, \boldsymbol{y}_{j'}) \geqslant 0$. It is called *positive definite* for the corresponding strict inequality.

In the following, we stick to *radial kernels*, i.e. kernel functions

$$k(\boldsymbol{y}, \boldsymbol{y}') := \varphi(\|\boldsymbol{y} - \boldsymbol{y}'\|)$$

with φ being a function $\varphi : [0, \infty) \to \mathbb{R}$. Typical examples are the *Gaussian kernel*, where $\varphi_G(r) := e^{-\epsilon^2 r^2}$, and the *Matérn kernel*, where

$$\varphi_M(r) := \frac{K_{\beta - \frac{d}{2}}(r) r^{\beta - \frac{d}{2}}}{2^{\beta - 1} \Gamma(\beta)}, \ \beta > \frac{d}{2}.$$

Here, K_ν is the modified Bessel function of the second kind of order ν and Γ is the gamma function.

A *native space* $\mathcal{N}_k(\Gamma)$ for a given symmetric and positive definite kernel function k is constructed by completion from the pre-Hilbert space $F_k(\Gamma) := \operatorname{span}\{k(\cdot, \boldsymbol{y}) \,|\, \boldsymbol{y} \in \Gamma\}$. It can be shown [Wen04] that $\mathcal{N}_k(\Gamma)$ is a RHKS for kernel k. The native space is equipped the induced norm from the inner product

$$(u, v)_{\mathcal{N}_k(\Gamma)} = \left(\sum_{j=1}^{N} \alpha_j \, k(\cdot, \boldsymbol{y}_j) \;,\; \sum_{k=1}^{N} \beta_k \, k(\cdot, \boldsymbol{y}'_k) \right)_{\mathcal{N}_k(\Gamma)} := \sum_{j=1}^{N} \sum_{k=1}^{N} \alpha_j \beta_k \, k(\boldsymbol{y}_j, \boldsymbol{y}'_k) \,.$$

As an example, the native space of a Matérn kernel with $\beta > d/2$ and $\Gamma = \mathbb{R}^d$ is the well-known Sobolev space $\mathcal{N}_{k_\beta}(\mathbb{R}^d) = H^\beta(\mathbb{R}^d)$.

In the following, we assume the given function u lies in the native space of a kernel k, i.e. $u \in \mathcal{N}_k(\Gamma)$. An appropriate finite-dimensional approximation subspace of $\mathcal{N}_k(\Gamma)$ can be introduced by choosing a finite set of sampling points

$$X = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\} \subset \Gamma \,.$$

Then, the finite-dimensional approximation subspace is

$$V = P_{k,X} := \operatorname{span}\{k(\cdot, \boldsymbol{y}) | \boldsymbol{y} \in X\} \subset \mathcal{N}_k(\Gamma) \,.$$

Example 1.4 (Interpolation in RKHS)**.** *Interpolation of a function $u \in \mathcal{N}_k(\Gamma)$ by a function in $P_{k,X}$ requires data $\{(\boldsymbol{y}_i, u(\boldsymbol{y}_i))\}_{i=1}^N$. Obviously, the interpolant $\mathcal{I}_{P_{k,X}} u$ becomes*

$$(\mathcal{I}_{k,X} u)(\boldsymbol{y}) := \sum_{j=1}^{N} \alpha_j k(\boldsymbol{y}, \boldsymbol{y}_j) \quad \forall \boldsymbol{y} \in \Gamma \,,$$

and we can find the coefficients $\{\alpha_j\}_{j=1}^N$, $\alpha_j \in \mathbb{R}$ by solving a system of linear equations with

$$\boldsymbol{A}_{P_{k,X}} \boldsymbol{\alpha} = \boldsymbol{u}, \qquad \boldsymbol{\alpha} := (\alpha_1 \ldots \alpha_N)^\top, \quad \boldsymbol{u} := (u(\boldsymbol{y}_1) \ldots u(\boldsymbol{y}_N))^\top \tag{1.10}$$

and the interpolation matrix

$$\boldsymbol{A}_{P_{k,X}} := \begin{pmatrix} k(\boldsymbol{y}_1, \boldsymbol{y}_1) & \ldots & k(\boldsymbol{y}_1, \boldsymbol{y}_N) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{y}_N, \boldsymbol{y}_1) & \ldots & k(\boldsymbol{y}_N, \boldsymbol{y}_N) \end{pmatrix} . \tag{1.11}$$

For symmetric and positive definite kernels, $\boldsymbol{A}_{P_{k,X}}$ becomes symmetric and positive definite and, hence, regular. △

Remark 1.5 (Best approximation and connection to machine learning)**.** *The interpolant* is *the best-approximation in the reproducing kernel Hilbert space $\mathcal{N}_k(\Gamma)$ [Wen04], i.e.*

$$\mathcal{I}_{P_{k,X}} u = \mathcal{B}_{P_k} u \,.$$

A Lavrentiev regularization [Lav67] leads to the linear system $(\boldsymbol{A}_{P_{k,X}} + \epsilon_{reg} \boldsymbol{I})\boldsymbol{\alpha} = \boldsymbol{u}$, transforming the interpolation problem to a kernel ridge regression *[Wen04, VSL+15] problem. Kernel ridge regression is a well-known technique in machine learning.*

Remark 1.6 (*h*- or *p*-refinement in kernel-based approxiamtion)**.** *Whether approximation in RKHS falls into the class of h- or p-refinement methods, depends on the applied kernel function. Matérn kernels with small β will lead to h-refinement, while Gaussian kernels applied in the approximation of a smooth function will be equivalent to p-refinement.*

Relation to achieved results

The multi-fidelity kernel ridge regression approach developed in Chapter 5 uses approximation in reproducing kernel Hilbert spaces / kernel ridge regression. Moreover Chapters 6 and 7 are concerned with the efficient solution of systems of linear equations with dense matrices as in equation (1.11).

1.5 Hierarchical approximation

In the following, we consider function approximation approaches that have in common that they rely on a hierarchy of finite dimensional approximation spaces.

1.5.1 Multi-level function approximation

Let $u : \Gamma \to \mathbb{R}$ be a function in a (separable) Hilbert space $\mathcal{F}$ with Γ being bounded. In *multi-level function approximation* we consider the approximation of u by a dense, nested sequence of finite dimensional subspaces

$$V_0 \subset V_1 \subset \cdots \subset V_\ell \subset \cdots \subset \mathcal{F}\,.$$

Each subspace V_ℓ is of dimension $N_\ell := \dim V_\ell$ and is spanned from a finite basis as

$$V_\ell = \operatorname{span}\left\{\phi_{\ell,j_\ell} \middle| j_\ell \in \{1,\ldots,N_\ell\}\right\}.$$

We consider approximations of the form

$$\mathcal{A}_\ell u := \sum_{j_\ell=1}^{N_\ell} \alpha_{\ell,j_\ell}\phi_{\ell,j_\ell}$$

on each level ℓ.

Example 1.5 (Multi-level finite elements for the Poisson problem)**.** *The use of multi-level function approximation is the base for multi-level finite element discretizations [ST03, HSS08], e.g., of the Poisson problem with homogeneous Dirichlet boundary conditions*

$$\begin{aligned} -\Delta u &= f \quad in\,\Gamma\,,\\ u &= 0 \quad on\,\partial\Gamma\,. \end{aligned}$$

The connection to approximation is as follows: We use a multi-level hierarchy of piecewise

linear polynomials as ansatz and test space for the problem given in weak form as

$$\int_\Gamma \nabla u(\boldsymbol{y}) \cdot \nabla v(\boldsymbol{y}) d\boldsymbol{y} = \int_\Gamma f(\boldsymbol{y}) v(\boldsymbol{y}) d\boldsymbol{y} \,,$$

where Γ *is a bounded domain that can be exactly given by a triangulation*

$$\mathcal{T}_0 := \{\tau_{0,1}, \ldots, \tau_{0,M_0}\} \,.$$

Then, we introduce the sequence $\{\mathcal{T}_\ell\}_{\ell=1}^{N_\ell}$ *of nested refinements of* $\mathcal{T}_0$*. For each such triangulation, we choose*

$$V_\ell := \{f \in C(\Gamma) | \forall \tau \in \mathcal{T}_\ell : f|_\tau \text{ is linear}\} \,,$$

i.e. globally continuous, piecewise linear polynomials on the triangulation. Thereby, we end up having a hierarchy of discretizations of the form

$$\boldsymbol{A}_\ell \boldsymbol{u}_\ell = \boldsymbol{f}_\ell \,,$$

where

$$a_{i_\ell, j_\ell} := \int_\Gamma \nabla \phi_{\ell, i_\ell} \cdot \nabla \phi_{\ell, j_\ell} d\boldsymbol{y}, \quad f_{i_\ell} := \int_\Gamma f(\boldsymbol{y}) \phi_{\ell, i_\ell} d\boldsymbol{y} \,.$$

To transfer solutions between different levels, prolongation and restriction operators are applied. △

Relation to achieved results

In Chapters 3 and 4, an algebraic approach (based on *algebraic multigrid*) is used to construct a multi-level finite element hierarchy similar to Example 1.5.

Example 1.6 (Multi-level kernel approximation)**.** *We can also apply multi-level function approximation in finite-dimensional subspaces of reproducing kernel Hilbert spaces [NSW99, Wen04]. In this case, we use* $\mathcal{F} = \mathcal{N}_k(\Gamma)$ *with a symmetric and positive definite kernel function* k*. Then, we introduce a nested sequence of meshfree point sets* $X_\ell := \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{N_\ell}\}$ *with* $N_0 < N_1 < \ldots < N_\ell$*. Each subspace* V_ℓ *becomes*

$$V_\ell := P_{k, X_\ell} = \operatorname{span}\{k(\cdot, \boldsymbol{y}) | \boldsymbol{y} \in X_\ell\} \,,$$

which induces a multi-level hierarchy of approximation spaces. Kernel interpolation / best approximation is done as discussed before. △

1.5.2 Approximation by hierarchical increments

As soon as we have introduced multiple levels of approximations, we can introduce the concept of *hierarchical increments*. For the sequence $V_0 \subset \cdots \subset V_\ell \subset \cdots \subset V_L$ of finite-dimensional approximation spaces, we can define incremental approximations

$$\Delta_\ell[\mathcal{A}]u = (\mathcal{A}_\ell - \mathcal{A}_{\ell-1})u \,,$$

where we start from $\Delta_0[\mathcal{A}] := \mathcal{A}_0$. This allows us to rewrite an approximation on level L as

$$\mathcal{A}_L = \sum_{\ell=0}^{L} \Delta_\ell[\mathcal{A}]u\,.$$

Such an incremental hierarchical approximation will be fundamental for the construction of *sparse* tensor-product approximations in Section 1.6.

Example 1.7 (Hierarchical approximation for piecewise linear polynomials on triangulations)**.** *We continue in the general setting of example 1.5, where we had the hierarchy of ansatz / approximation spaces*

$$V_\ell := \{f \in C(\Gamma) | \forall \tau \in \mathcal{T}_\ell : f|_\tau \text{ is linear}\}$$

for a sequence of nested triangulations.

Approximation in this space is typically done by L^2 projection. That is, the task is to find a $\mathcal{P}_{V_\ell} u_\ell \in V_\ell$ such that

$$\mathcal{P}_{V_\ell} u := \operatorname*{argmin}_{u_\ell \in V_\ell} \|u_\ell - u\|_{L^2(\Gamma)}\,.$$

With the L^2-orthogonal projection $\mathcal{P}_{V_\ell} : L^2(\Gamma) \to V_\ell$, we can introduce the incremental projection $\Delta_\ell[\mathcal{P}]$ with

$$\Delta_\ell[\mathcal{P}] := \mathcal{P}_{V_\ell} - \mathcal{P}_{V_{\ell-1}}, \quad \ell \in \mathbb{N}, \quad \Delta_0[\mathcal{P}] := \mathcal{P}_{V_0}.$$

Then, we can incrementally compute the approximation $\mathcal{P}_{V_L} u$ of u on level L by

$$\mathcal{P}_{V_L} u := \sum_{\ell=0}^{L} \Delta_\ell[\mathcal{P}]u\,.$$

Due to the properties of the spaces V_ℓ we can further introduce increment *or* complementary *spaces W_ℓ such that*

$$V_\ell = V_{\ell-1} \overset{\perp}{\oplus} W_\ell, \quad V_0 = W_0, \quad V_{\ell-1} \cap W_\ell = \{0\}, \quad W_\ell = \operatorname{span}\{\phi_{\ell,j_\ell} | j_\ell \in \nabla_\ell\}\,,$$

with ∇_ℓ the set of basis indices that differ between level $\ell-1$ and ℓ. Thereby, we can recursively decompose a space V_L as

$$V_L = \bigoplus_{\ell=0}^{L} W_\ell, \quad W_0 := V_0$$

and $\Delta_\ell[\mathcal{P}]$ is a projection onto W_ℓ, i.e. $\Delta_\ell[\mathcal{P}] : V \to W_\ell$. △

Relation to achieved results

The sparse second moment analysis in Chapter 4 specifically makes use of hierarchical increments from an *algebraic* multi-level construction.

Remark 1.7 (Hierarchical approximation in RKHS)**.** *As in the previous example, we can (in extension of Example 1.6) build an incremental interpolation in reproducing kernel Hilbert*

spaces, as proposed in [NSW99]. However, as stated in this reference, we cannot associate increment spaces *such as W_ℓ to this approximation, since the approximations on two consecutive levels cannot be brought in a form that they are direct sums of subspaces.*

1.5.3 Multi-level frames

An extension to the multi-level approximation is given by the approximation with multi-level *frames*. Again, we start from a sequence of nested finite-dimensional approximation spaces

$$V_0 \subset \cdots \subset V_\ell \subset \cdots \subset V_L \subset \mathcal{F}\,,$$

with $V_\ell := \operatorname{span}\{\phi_{\ell,1}, \ldots, \phi_{\ell,N_\ell}\}$ and $\mathcal{F}$ is now assumed to be a separable Hilbert function space with its dual space $\mathcal{F}'$. We introduce a collection Φ_L of all basis functions up to level L

$$\Phi_L := \left\{\phi_{\ell,j_\ell} \middle| j_\ell \in \{1, \ldots, N_\ell\}, \ell \in \{0, \ldots, L\}\right\}.$$

Obviously, Φ_L is no longer a basis, but a *generating system*. Moreover, following [HSS08], Φ_L is called a *frame* for $\mathcal{F}$, if it holds

$$c_1 \|f\|^2_{\mathcal{F}'} \leqslant \sum_{\ell,j_\ell} |\langle f, \phi_{\ell,j_\ell}\rangle|^2 \leqslant c_2 \|f\|^2_{\mathcal{F}'} \quad \forall f \in \mathcal{F}'\,. \tag{1.12}$$

Function approximation in multi-level frames is the task to find (non-unique) coefficients α_{ℓ,j_ℓ} for the representation

$$\mathcal{A}_{F,\Phi_L} u = \sum_{\ell=0}^{L} \sum_{j_\ell=1}^{N_\ell} \alpha_{\ell,j_\ell} \phi_{\ell,j_\ell}$$

such that $\mathcal{A}_{F,\Phi_L} u \approx u$.

Example 1.8 (L^2-projection onto frames)**.** *Continuing Example 1.1, we can do best approximation of a function $u \in L^2(\Gamma)$ with respect to a frame by the L^2 projection*

$$\mathcal{P}_{F,\Phi_L} u = \sum_{\ell=0}^{L} \sum_{j_\ell=1}^{N_\ell} \alpha_{\ell,j_\ell} \phi_{\ell,j_\ell}\,.$$

The coefficients α_{ℓ,j_ℓ} are computed by solving the system of linear equations

$$\sum_{\ell=0}^{L} \sum_{j_\ell=1}^{N_\ell} \langle \phi_{\ell,j_\ell}, \phi_{\ell',i_{\ell'}}\rangle \alpha_{\ell,j_\ell} = \langle u, \phi_{\ell',i_{\ell'}}\rangle \quad \forall \ell' \in \{0, \ldots, L\}\ i_{\ell'} \in \{1, \ldots, N_\ell\}\,.$$

Note that this system of linear equations is not uniquely solvable. However, we can find solutions that are compatible with (1.12) *by using an iterative conjugate gradient solver, as long as we do not use an initial guess that is in the kernel of the system matrix.* △

Relation to achieved results

In Chapters 3 and 4, frames are constructed from an algebraic multi-level approach that is based on algebraic multigrid. Chapter 3 discusses the application of such algebraic frames in context of optimal iterative linear solvers, whereas the algebraic sparse tensor product construction in Chapter 4 applies algebraic frames for an application from uncertainty quantification.

1.6 Sparse tensor-product approximation

At the end of Example 1.3, we observed that standard tensor product approximation can be very expensive in terms of the required amount of function evaluations. To reduce the amount of function evaluations for large dimensions d, we can introduce *sparse* tensor-product approximations.

In the following, we mostly (i.e. with the exception of Section 1.6.4) discuss the case of $\Gamma := \Gamma^{(1)} \times \ldots \times \Gamma^{(d)}$. For each $\Gamma^{(i)}$ we have a Hilbert function space $\mathcal{F}^{(i)}$ and aim at approximating functions $u \in \mathcal{F}^{(1)} \otimes \cdots \otimes \mathcal{F}^{(d)}$. Moreover, for each $\mathcal{F}^{(i)}$, we have a nested sequence of finite-dimensional approximation spaces $V_{\ell_i}^{(i)}$ with non-negative levels ℓ_i.

1.6.1 Main approach

Recall from Section 1.4 that standard tensor product approximation uses approximations in the space $V_{\ell_1}^{(1)} \otimes \cdots \otimes V_{\ell_d}^{(d)}$ with $\ell_1 = \ldots = \ell_d = L$. Based on the incremental approximations Δ_{ℓ_i} from Section 1.5.2, we could have defined standard tensor product approximation by

$$\mathcal{A}_{T_L} u := \sum_{|\boldsymbol{\ell}|_\infty \leqslant L} \mathcal{A}_{T\Delta_{\boldsymbol{\ell}}} u := \sum_{|\boldsymbol{\ell}|_\infty \leqslant L} \left(\Delta_{\ell_1}[\mathcal{A}_{V^{(1)}}] \otimes \cdots \otimes \Delta_{\ell_d}[\mathcal{A}_{V^{(d)}}]\right) u \tag{1.13}$$

with $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_d)$ and $|\boldsymbol{\ell}|_\infty := \max_i |\ell_i|$. Hence, $\mathcal{A}_{T\Delta_{\boldsymbol{\ell}}}$ is a tensor product over incremental approximations on levels ℓ_i. Note that summation over a multi-index $\boldsymbol{\ell} \in \mathbb{N}^d$ corresponds to d nested sums over the element-wise indices of the multi-index.

Sparse tensor product approximation [HSS08, GH13, CD15] is given by

$$\mathcal{A}_{S_L} u := \sum_{|\boldsymbol{\ell}|_1 \leqslant L} \left(\Delta_{\ell_1}[\mathcal{A}_{V^{(1)}}] \otimes \cdots \otimes \Delta_{\ell_d}[\mathcal{A}_{V^{(d)}}]\right) u \tag{1.14}$$

with $|\boldsymbol{\ell}|_1 := \sum_{i=1}^d |\ell_i|$. If we denote by $\mathcal{A}_{S_L}^{(i)}$ the sparse approximation operator that is defined on a finite-dimensional subspace of $\mathcal{F}^{(1)} \otimes \cdots \otimes \mathcal{F}^{(i)}$, $i \leqslant d$, we can [HPS13] rewrite (1.14) to be recursively given by

$$\mathcal{A}_{S_L}^{(i)} u = \sum_{\ell_i=0}^{L} \left(\Delta_{\ell_i}[\mathcal{A}_{V^{(i)}}] \otimes \mathcal{A}_{S_L}^{(i-1)}\right) u\,,$$

observing that it holds $\mathcal{A}_{S_L} = \mathcal{A}_{S_L}^{(d)} u$.

Example 1.9 (Sparse grid interpolation)**.** *Standard* sparse grid interpolation *[Smo63, BG04] for functions, e.g., of the form* $u : [-1,1]^d \to \mathbb{R}$ *corresponds to sparse tensor product interpolation constructed from* d *piecewise linear approximation spaces* $V_{\ell_i}^{(i)} := PW_{N_L}$*, cf.* (1.9)*. That is, the sparse grid interpolation formula for a given interpolation level* L *is*

$$\mathcal{I}_{S_L,X} u := \sum_{|\boldsymbol{\ell}|_1 \leqslant L} \left(\Delta_{\ell_1}[\mathcal{I}_{V^{(1)},X^{(1)}}] \otimes \cdots \otimes \Delta_{\ell_d}[\mathcal{I}_{V^{(d)},X^{(d)}}] \right) u \,,$$

Comparing this interpolation formula to full tensor product interpolation by piecewise linear functions, cf. Example 1.3, we observe that the sparse approach needs much less function evaluations than the full tensor product approach. Error estimates for the full tensor product interpolation require bounded derivatives in each coordinate direction. To obtain roughly the same estimates for sparse grid interpolation, we further have to require boundedness of the mixed derivatives of u*. Often, we assume*

$$u \in \mathcal{W}_d^p(\Gamma) := \left\{ f : \Gamma \to \mathbb{R} \;\middle|\; \left\| \frac{\partial^{|\boldsymbol{s}|_1} f}{\partial y_1^{s_1} \cdots \partial y_d^{s_d}} \right\|_\infty < \infty,\ |\boldsymbol{s}|_\infty \leqslant p \right\} .$$

△

Example 1.10 (Sparse grid stochastic collocation)**.** *The well-known sparse grid stochastic collocation for the solution of partial differential equations with random input [NTW08] corresponds to sparse tensor-product interpolation using univariate polynomial approximation spaces* $V_{\ell_i}^{(i)} := P_{N_\ell}$ *and e.g. Clenshaw-Curtis abscissas, cf.* (1.8)*.* △

Example 1.11 (Sparse tensor product kernel approximations)**.** *In, e.g., [DGLU15, RW17], sparse function approximation is achieved by building sparse tensor product interpolation from univariate approximation spaces* $V_{\ell_i}^{(i)} := \operatorname{span}\{k(\cdot, \boldsymbol{y}_i) | \boldsymbol{y}_i \in X_{\ell_i}\}$ *with* X_{ℓ_i} *uniformly distributed points on a closed interval.* △

1.6.2 Generalized sparse tensor product approximation

A generalization of (1.14) can be obtained by approximation with respect to a *downward closed* index set $\mathcal{I} \subset \mathbb{N}_0^d$. $\mathcal{I}$ is called downward closed [CD15], if it holds

$$\boldsymbol{\ell} \in \mathcal{I}, \tilde{\boldsymbol{\ell}} \leqslant \boldsymbol{\ell} \Rightarrow \tilde{\boldsymbol{\ell}} \in \mathcal{I} \,,$$

with a component-wise comparison "$\leqslant$". Then, generalized sparse tensor product approximation is given by

$$\mathcal{A}_{S_\mathcal{I}} u := \sum_{\boldsymbol{\ell} \in \mathcal{I}} \beta_{\boldsymbol{\ell}} \left(\Delta_{\ell_1}[\mathcal{A}_{V^{(1)}}] \otimes \cdots \otimes \Delta_{\ell_d}[\mathcal{A}_{V^{(d)}}] \right) u \tag{1.15}$$

with coefficients $\beta_{\boldsymbol{\ell}}$ defined by

$$\beta_{\boldsymbol{\ell}} := \sum_{\boldsymbol{z} \in \{0,1\}^d} (-1)^{|\boldsymbol{z}|_1} \chi_\mathcal{I}(\boldsymbol{\ell} + \boldsymbol{z}) \,, \quad \chi_\mathcal{I}(\boldsymbol{\ell} + \boldsymbol{z}) := \begin{cases} 1, & \text{if } (\boldsymbol{\ell} + \boldsymbol{z}) \in \mathcal{I}, \\ 0, & \text{else.} \end{cases} \tag{1.16}$$

Notice that index set $\mathcal{I}$ can, e.g., be found by the solution of a *Knapsack* optimization problem [BG04].

1.6.3 Sparse grid combination technique

The *sparse grid combination technique* [GSZ92] is strongly related to sparse grid tensor product approximations. With the definitions from the previous section, it is given by

$$\begin{aligned}\mathcal{A}_{C_L} u :&= \sum_{i=0}^{d-1} (-1)^i \binom{d-1}{i} \sum_{|\boldsymbol{\ell}|_1 = L-i} \left(\mathcal{A}_{V_{\ell_1}^{(1)}} \otimes \cdots \otimes \mathcal{A}_{V_{\ell_d}^{(d)}} \right) u \\ &= \sum_{i=0}^{d-1} (-1)^i \binom{d-1}{i} \sum_{|\boldsymbol{\ell}|_1 = L-i} \mathcal{A}_{T_{\boldsymbol{\ell}}} u \,.\end{aligned}$$

Each summand $\mathcal{A}_{T_{\boldsymbol{\ell}}} u$ is an *anisotropic* full tensor product approximation of the underlying function u, i.e.

$$\mathcal{A}_{T_{\boldsymbol{\ell}}} u := \sum_{\boldsymbol{\ell}' \leqslant \boldsymbol{\ell}} \mathcal{A}_{T \Delta_{\boldsymbol{\ell}'}} u \,,$$

cf. (1.13). The summands are then combined in an appropriate linear combination to a *sparse grid*-type approximation. A *generalized* version of the sparse grid combination technique is given by

$$\mathcal{A}_{C_{\mathcal{I}}} u := \sum_{\boldsymbol{\ell} \in \mathcal{I}} \beta_{\boldsymbol{\ell}} \mathcal{A}_{T_{\boldsymbol{\ell}}} u$$

with $\beta_{\boldsymbol{\ell}}$ from equation (1.16).

Example 1.12 (Sparse grid interpolation using the combination technique)**.** *We continue Example 1.9. Using the sparse grid combination technique, we obtain the interpolation rule*

$$\mathcal{I}_{C_L, X} u := \sum_{i=0}^{d-1} (-1)^i \binom{d-1}{i} \sum_{|\boldsymbol{\ell}|_1 = L-i} \left(\mathcal{I}_{V_{\ell_1}^{(1)}, X_{\ell_1}^{(1)}} \otimes \cdots \otimes \mathcal{I}_{V_{\ell_d}^{(d)}, X_{\ell_d}^{(d)}} \right) u \,. \tag{1.17}$$

This interpolation rule is identical to the classical sparse grid interpolation rule $\mathcal{I}_{S_L, X} u$ [DS89, BG04]. △

Relation to achieved results

In Chapter 5, the sparse grid combination technique is combined with the approximation in reproducing kernel Hilbert spaces / kernel ridge regression in order to introduce a multi-fidelity machine learning approach for quantum chemistry calculations. Chapter 4 constructs an *algebraic* sparse grid combination technique to solve a model problem from second moment analysis of elliptic problems.

1.6.4 Multi-index approximation

Multi-index approximation [HANTT16] being developed out of Monte-Carlo and multi-level Monte-Carlo methods, considers in its function approximation version (i.e. multi-index collocation) an alternative view of the sparse grid combination technique, starting from function approximations

$$\mathcal{A}_{\boldsymbol{\ell}} u$$

with $\boldsymbol{\ell} \in \mathbb{N}_0^d$ a multi-index. Notice Remark 1.8 for an interpretation of such function approximations.

Using $\mathcal{A}_{\boldsymbol{\ell}}$, literature on multi-index approximation [HANT16, HANTT16] introduces *first-order difference operators* Δ_i along directions $1 \leqslant i \leqslant d$ as

$$\Delta_i[\mathcal{A}_{\boldsymbol{\ell}} u] := \begin{cases} \mathcal{A}_{\boldsymbol{\ell}} u - \mathcal{A}_{\boldsymbol{\ell}-\boldsymbol{e}_i} u, & \text{if } \ell_i > 0\,, \\ \mathcal{A}_{\boldsymbol{\ell}} u, & \text{if } \ell_i = 0\,, \end{cases} \tag{1.18}$$

with unit vectors $\boldsymbol{e}_i$. In d dimensions, we obtain the *first-order mixed difference operator* Δ via

$$\Delta[\mathcal{A}_{\boldsymbol{\ell}} u] := \left(\bigotimes_{i=1}^{d} \Delta_i\right)[\mathcal{A}_{\boldsymbol{\ell}} u] = \Delta_1 \left(\bigotimes_{i=2}^{d} \Delta_i\right)[\mathcal{A}_{\boldsymbol{\ell}} u] = \Delta_d \left(\bigotimes_{i=1}^{d-1} \Delta_i\right)[\mathcal{A}_{\boldsymbol{\ell}} u]\,.$$

For a general downward closed index set $\mathcal{I}$, we obtain the multi-index approximation $\mathcal{A}_{M_{\mathcal{I}}} u$ by

$$\mathcal{A}_{M_{\mathcal{I}}} u = \sum_{\boldsymbol{\ell} \in \mathcal{I}} \beta_{\boldsymbol{\ell}} \Delta[\mathcal{A}_{\boldsymbol{\ell}} u]\,, \tag{1.19}$$

with $\beta_{\boldsymbol{\ell}}$ from equation (1.16).

Remark 1.8 (Equivalence to generalized sparse grid combination technique)**.** *The generalized sparse grid combination technique and multi-index approximation (by collocation) following* (1.19)*, are — up to the choice of the employed approximations $\mathcal{A}_{\boldsymbol{\ell}} u$ — the* same *approximations. For $\mathcal{A}_{\boldsymbol{\ell}} = \mathcal{A}_{T_{\boldsymbol{\ell}}}$, i.e. if the multi-index describes the anisotropic level parameter of a tensor-product approximation, both approximations are identical. As discussed in [HANTT16], $\mathcal{A}_{\boldsymbol{\ell}}$ could can be any type of approximation with d level parameters $\{\ell_i\}_{i=1}^d$. That is, it would not be restricted to tensor product approximations. However, to the best of the author's knowledge, all approximation problems discussed in literature on multi-index collocation are indeed tensor product problems.*

Relation to achieved results

The multi-fidelity machine learning approach discussed in Chapter 5 can be interpreted as multi-index approximation in reproducing kernel Hilbert spaces with regularization.

1.7 Low-rank approximation

In this section, we are interested in the approximation of functions $\boldsymbol{A}$ of type

$$\boldsymbol{A} : \mathcal{I}^{(1)} \times \cdots \times \mathcal{I}^{(\mathfrak{d})} \to \mathbb{R}\,, \tag{1.20}$$

where each $\mathcal{I}^{(i)}$ is an index set $\mathcal{I}^{(i)} := \{1, \dots, N_i\}$. For $\mathfrak{d} = 1$ we call $\boldsymbol{A}$ *vector*, for $\mathfrak{d} = 2$ we call $\boldsymbol{A}$ *matrix* and for $\mathfrak{d} > 2$ we call $\boldsymbol{A}$ *tensor of order* $\mathfrak{d}$.

The approximation of $\boldsymbol{A}$ can become a high-dimensional approximation problem if either $\mathfrak{d}$ is large or if $\mathfrak{d}$ is small but the index sets $I^{(i)}$ represent objects in a higher-dimensional space.

Example 1.13 (Tensor-approximation of parametric PDEs)**.** *We can consider the approximation of a parametric PDE*

$$\mathcal{L}(\boldsymbol{a}(\boldsymbol{y}, \boldsymbol{x}))u(\boldsymbol{y}, \boldsymbol{x}) = f[\boldsymbol{b}(\boldsymbol{y}, \boldsymbol{x})](\boldsymbol{y}, \boldsymbol{x}) \qquad \text{in } \Gamma \times \bar{\mathcal{D}}$$

as a purely discrete tensor approximation problem [BG15]. To this end, we introduce a spatial discretization with $N_{\mathcal{D}}$ grid points in a set $X_{\mathcal{D}}$ with indices $\mathcal{I}_{\mathcal{D}}$. Moreover, we assume $\Gamma = \Gamma^{(1)} \times \cdots \times \Gamma^{(d)}$ with $\Gamma^{(i)}$ closed intervals in $\mathbb{R}$. In each dimension i, we, e.g., use a uniform grid with $N_{\Gamma^{(i)}}$ points in sets $X_{\Gamma^{(i)}}$ and indices $\mathcal{I}_{\Gamma^{(i)}}$. Formally, we can then represent the parametric solution $u(\boldsymbol{y}, \boldsymbol{x})$ to arbitrary precision (for $N_{\mathcal{D}}$, $N_{\Gamma^{(i)}}$ large enough) by the mapping

$$\begin{aligned} \boldsymbol{A} : \mathcal{I}_{\mathcal{D}} \times \mathcal{I}_{\Gamma^{(1)}} \times \mathcal{I}_{\Gamma^{(2)}} \times \cdots \times \mathcal{I}_{\Gamma^{(d)}} &\to \mathbb{R}\,, \\ (\boldsymbol{x}_{j_{\mathcal{D}}}, y_{j_1}, \dots, y_{j_d}) &\mapsto u(\boldsymbol{x}_{j_{\mathcal{D}}}, (y_{j_1}, \dots, y_{j_d}))\,. \end{aligned}$$

Hence, we represent the parametric solution as a $(d+1)$-fold tensor. Obviously, the number of non-parametric PDE evaluations to obtain each entry of this tensor is computationally intractable. Instead, we consider an approximation of $\boldsymbol{A}$, which can drastically reduce the number of PDE evaluations. △

Example 1.14 (Matrix approximation for function interpolation in RKHS)**.** *In Section 1.4, we discussed function interpolation in RKHS by means of radial kernel functions k. To interpolate a given function, we have the set $X = \{\boldsymbol{y}_1, \dots, \boldsymbol{y}_N\} \subset \Gamma \subset \mathbb{R}^d$ of N abscissas and have to solve a linear system of the form*

$$\boldsymbol{A}\boldsymbol{\alpha} = \boldsymbol{f} \tag{1.21}$$

with $\boldsymbol{A} := (k(\boldsymbol{y}_i, \boldsymbol{y}_{i'}))_{i,i'=1}^{N}$.

For an efficient solution of equation (1.21) for large N, we are interested to approximate the system matrix $\boldsymbol{A}$. Using $\mathcal{I}^{(1)} = \mathcal{I}^{(2)} := \{1, \dots, N\}$, we can cast $\boldsymbol{A}$ to the general setting of equation (1.20) with $\mathfrak{d} = 2$ as

$$\begin{aligned} \boldsymbol{A} : \mathcal{I}^{(1)} \times \mathcal{I}^{(2)} &\to \mathbb{R}\,, \\ (i, i') &\mapsto k(\boldsymbol{y}_i, \boldsymbol{y}_{i'})\,. \end{aligned}$$

The approximation of a matrix is not necessarily understood as higher-dimensional problem. However, we actually approximate a finite number of evaluations of k, which is a function that maps from the $2d$-dimensional space $\Gamma \times \Gamma$ to the real numbers. Therefore, we end up approximating a higher-dimensional object. △

Relation to achieved results

The multi-fidelity kernel ridge regression approach from Chapter 5 involves the solution of a (regularized) linear system of equations with a system matrix as in Example 1.14. Chapters 6 and 7 discuss the efficient parallelization and implementation of approximation techniques for Example 1.14.

This section surveys techniques for approximating two-fold tensors, i.e. matrices, by means of *low-rank approximation*. Remark 1.9 will motivate, how to extend the low-rank approximation of matrices to tensor approximation by the use of *tensor formats*.

Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ be a real-valued matrix. For simplicity of the presentation, we assume $m \leqslant n$. Moreover, we have $r \in \mathbb{N}_{>0}$, such that $r < m$ and $\|\cdot\|$ a matrix norm. In *low-rank approximation* [Mar08], we aim at finding a matrix $\boldsymbol{A}_r$ that fulfills the minimization problem

$$\boldsymbol{A}_r := \underset{\tilde{\boldsymbol{A}} \in \mathbb{R}^{m \times n}}{\operatorname{argmin}} \ \|\boldsymbol{A} - \tilde{\boldsymbol{A}}\|, \quad \text{such that } \operatorname{rank}(\tilde{\boldsymbol{A}}) \leqslant r\,. \tag{1.22}$$

Hence, low-rank approximations are best approximations by rank-r matrices with respect to the $\|\cdot\|$ norm.

1.7.1 Singular value decomposition

Following e.g. [HH94], the decomposition of a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ of the form

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$$

with $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ being orthogonal matrices is called *singular value decomposition (SVD)*. The matrix $\boldsymbol{\Sigma}$ is a diagonal matrix with non-negative diagonal entries $\{\sigma_i\}_{i=1}^{\min\{m,n\}}$, the *singular values*, in the upper left part of the matrix. We will assume that the singular values are sorted following a decreasing order. The SVD exists for any matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and it can be computed by a method related to the QR algorithm in $O(\max\{m,n\}^3)$ operations [SB05].

With $\boldsymbol{U} = (\boldsymbol{u}_1|\ldots|\boldsymbol{u}_m)$ and $\boldsymbol{V} = (\boldsymbol{v}_1|\ldots|\boldsymbol{v}_n)$ the approximation

$$\boldsymbol{A} \approx \boldsymbol{A}_{svd,r} := \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\top$$

is [Mar08] the low-rank approximation as defined in (1.22) with respect to the Frobenius norm $\|\cdot\|_F$. Unfortunately, always, the *full* matrix $\boldsymbol{A}$ has to be computed in order to build the approximation $\boldsymbol{A}_{svd,r}$. Even a partial SVD [GTY97] will require the evaluation of all matrix entries of $\boldsymbol{A}$ [Beb00].

1.7.2 Adaptive cross approximation

The *Adaptive Cross Approximation* [Beb00] (with partial pivoting), cf. Algorithm 1 and Chapter 6, promises to overcome the high computational cost of the SVD. It uses successive rank-1

Algorithm 1 Adaptive cross approximation (ACA) for $\boldsymbol{A} \in \mathbb{R}^{m \times n}$

function COMPUTE_ADAPTIVE_CROSS_APPROXIMATION($\boldsymbol{A}$, ϵ)
  $k_{max} \leftarrow k$
  for $r = 1, 2, \ldots, k$ **do**
    $\hat{\boldsymbol{u}}_r = \boldsymbol{A}_{1:m,j_r} - \sum_{l=1}^{r-1} \boldsymbol{u}_l (\boldsymbol{v}_l)_{j_r}$, ▷ Col. index j_r depending on implementation
    $\boldsymbol{u}_r = (\hat{\boldsymbol{u}}_{i_r})^{-1} \hat{\boldsymbol{u}}_r$, with $|(\hat{\boldsymbol{u}}_r)_{i_r}| = \|\hat{\boldsymbol{u}}_r\|_\infty$ ▷ Row index i_r given as pivot position
    $\boldsymbol{v}_r = (A_{i_r,1:n})^\top - \sum_{l=1}^{r-1} (\boldsymbol{u}_l)_{i_r} \boldsymbol{v}_l$
    if $\left(\|\boldsymbol{u}_r\|_2 \|\boldsymbol{v}_r\|_2 \leqslant \frac{\epsilon(1.0-\eta)}{1.0+\epsilon} \left\| \sum_{l=1}^{r} \boldsymbol{u}_l \boldsymbol{v}_l \right\|_F \right)$ **then** ▷ Stopping criterion
      $k_{max} \leftarrow r$ ▷ k_{max} is adaptively found rank
      stop loop
  $U \leftarrow (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{k_{max}})$
  $V \leftarrow (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k_{max}})$
  return U, V

updates to build an approximation of rank $k_{max}(\epsilon)$ of the form

$$\boldsymbol{A} \approx \boldsymbol{A}_{aca,\epsilon} = \sum_{i=1}^{k_{max}(\epsilon)} \boldsymbol{u}_i \boldsymbol{v}_i^\top$$

with $\boldsymbol{u}_i$, $\boldsymbol{v}_i$ being defined in Algorithm 1. Approximations to $\boldsymbol{A}$ are computed for a fixed tolerance ϵ with respect to the Frobenius norm. The tolerance influences the rank $k_{max}(\epsilon)$ of the approximation. Since ACA only evaluates some rows and columns of $\boldsymbol{A}$, it allows to build approximations of $\boldsymbol{A}$ in an efficient way.

Example 1.15 (Approximation of kernel matrices by ACA)**.** *We continue Example 1.14. Hence, we discuss the approximation of the system matrix showing up in interpolation in RKHS. It is known from [Beb00] that ACA (with full pivoting) converges exponentially for smooth kernels k. This holds for example in the case of the Gaussian kernel $k(\boldsymbol{y}, \boldsymbol{y'}) := e^{-\epsilon^2 \|\boldsymbol{y}-\boldsymbol{y'}\|^2}$, i.e. the matrix $\boldsymbol{A} := (e^{-\epsilon^2 \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2})_{i,j=1}^N$.*

However, many radial kernel functions, such as the important example of kernels of the Matérn class, cf. Section 1.4, are not smooth at the diagonal $k(\boldsymbol{y}, \boldsymbol{y})$. In these cases, this favorable error decay is no longer present. Hierarchical matrices *[Hac15] introduce a matrix approximation that allows to overcome this limitation. $\mathcal{H}$-matrices will be discussed in Section 1.7.3.* △

Relation to achieved results

In Chapters 6 and 7, ACA is applied in context of *hierarchical matrices* for the approximation of kernel matrices, cf. Example 1.15.

Remark 1.9 (Tensor approximation)**.** *All low-rank approximation techniques discussed so far were introduced for* matrices. *We now sketch how to extend this idea to* order-$\mathfrak{d}$ tensors,

cf. [BG15]. Recall that such a tensor is of the form

$$\boldsymbol{A} : \mathcal{I}^{(1)} \times \cdots \times \mathcal{I}^{(\mathfrak{d})} \to \mathbb{R}\,. \tag{1.23}$$

From a notation point of view, we can also write $\boldsymbol{A} \in \mathbb{R}^{\mathcal{I}}$ *with* $\mathcal{I} := \mathcal{I}_1 \times \cdots \times \mathcal{I}_{\mathfrak{d}}$*. Collecting all dimensions in a set* $X_{\mathfrak{d}} := \{1, \ldots, \mathfrak{d}\}$*, we can group dimensions in subsets* $\mathfrak{t} \subset X_{\mathfrak{d}}$ *with complements* $\bar{\mathfrak{t}} := X_{\mathfrak{d}} \backslash \mathfrak{t}$ *giving rise to the notation* $\mathcal{I}_{\mathfrak{t}} := \bigtimes_{i \in \mathfrak{t}} \mathcal{I}^{(i)}$*. A* matricisation *with respect to* $\mathfrak{t} \subset X_{\mathfrak{d}}$ *is a function*

$$\mathfrak{M} : \mathbb{R}^{\mathcal{I}} \to \mathbb{R}^{\mathcal{I}_{\mathfrak{t}}} \otimes \mathbb{R}^{\mathcal{I}_{\bar{\mathfrak{t}}}}$$

that splits an order-$\mathfrak{d}$ *tensor into the tensor product of two tensors with underlying index sets* $\mathcal{I}_{\mathfrak{t}}$ *and* $\mathcal{I}_{\bar{\mathfrak{t}}}$*. The tensor product* $\mathbb{R}^{\mathcal{I}_{\mathfrak{t}}} \otimes \mathbb{R}^{\mathcal{I}_{\bar{\mathfrak{t}}}}$ *can be understood as the set of all* matrices *of size* $\mathbb{R}^{|\mathcal{I}_{\mathfrak{t}}| \times |\mathcal{I}_{\bar{\mathfrak{t}}}|}$ *after linearization of the index sets* $\mathcal{I}_{\mathfrak{t}}$ *and* $\mathcal{I}_{\bar{\mathfrak{t}}}$*. Hence, matricisation of a tensor is a linearization and reordering of the index sets by dimension.*

Matriciation translates tensors into matrices. Thereby, it becomes possible to apply low-rank approximation techniques to tensors. To get efficient low-rank approximations of tensors, tensors are recursively subdivided as exemplified in Figure 1.9 following [BG15] for a five-fold tensor. The sets correspond to the subsets $\mathfrak{t}$ *discussed before. The type of structure of this subdivision is called* tensor format*. The decomposition shown in Figure 1.9 corresponds to a* hierarchical *tensor format. Roughly speaking, low-rank approximations via matriciations are done for each dimension split.*

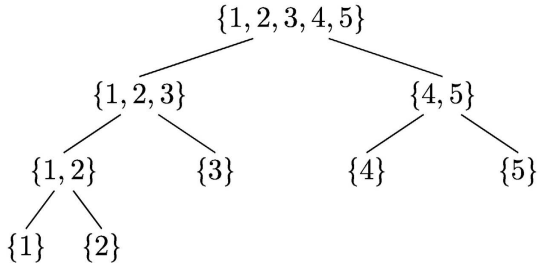


Figure 1.9: Hierarchical tensor formats are based on a dimension-wise decomposition of the tensor and its associated index sets [BG15].

1.7.3 Hierarchical matrices

Since Examples 1.14 and 1.15 are fundamental for this thesis, we would like to invest some more time in the discussion of an approximation of matrices of type

$$\boldsymbol{A} := (k(\boldsymbol{y}_i, \boldsymbol{y}_j))_{i,j=1}^{N}\,,$$

with points $\boldsymbol{y}_i$ in a set $X := \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\} \subset \Gamma$. As discussed in Example 1.15, some kernel functions k are not smooth on the diagonal, however they are smooth away from the diagonal. This is formalizes by the notion *asymptotically smooth*. A kernel k is asymptotically smooth,

if we have constants $c_1, c_2 \in \mathbb{R}_{>0}$ such that

$$|\partial_{\boldsymbol{y}}^{\boldsymbol{\alpha}} \partial_{\boldsymbol{y}'}^{\boldsymbol{\beta}} k(\boldsymbol{y}, \boldsymbol{y}')| \leqslant c_1 \frac{(|\boldsymbol{\alpha}| + |\boldsymbol{\beta}|)!}{(c_2 \|\boldsymbol{y} - \boldsymbol{y}'\|)^{|\boldsymbol{\alpha}|+|\boldsymbol{\beta}|}} |k(\boldsymbol{y}, \boldsymbol{y}')|$$

for arbitrary $\boldsymbol{y}, \boldsymbol{y}' \in \Gamma$ with $\boldsymbol{y} \neq \boldsymbol{y}'$ and all multi-indices $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{N}_0^d$.

Matrices $\boldsymbol{A}$ constructed from asymptotically smooth kernels k can be approximated by a combination of a hierarchical decomposition of $\boldsymbol{A}$ and ACA in $O(N \log N)$ complexity (for fixed target accuracy ϵ). This type of approximation is called *hierarchical matrix* of *$\mathcal{H}$-matrix* [BGH03, Hac15].

Using a hierarchical clustering of the points in X, $\mathcal{H}$-matrices can restructure a given input matrix $\boldsymbol{A}$ of the above form such that one can distinguish two classes of matrix blocks. *Admissible* matrix blocks are blocks with entries $k(\boldsymbol{y}, \boldsymbol{y}')$ such that $\boldsymbol{y}$ and $\boldsymbol{y}'$ are far away from each other. *Inadmissible* matrix blocks are blocks with entries $k(\boldsymbol{y}, \boldsymbol{y}')$ such that $\boldsymbol{y}$ and $\boldsymbol{y}'$ are close to each other. Admissible matrix blocks only contain evaluations of the kernel k that are away from the diagonal, hence k is smooth in this regime. Therefore, cf. Example 1.15, an approximation of an admissible block can efficiently be done by ACA. In contrast, inadmissible blocks contain evaluations of k in the non-smooth regime close to the diagonal. Therefore, no approximation is applied to inadmissible blocks. As long as we have a controlled, small amount of small inadmissible blocks, $\mathcal{H}$-matrices are a very efficient mean to approximate the above kernel matrices. For further details on $\mathcal{H}$-matrices, see Chapters 6 and 7.

Remark 1.10 (ACA and $\mathcal{H}$-matrics)**.** *The original $\mathcal{H}$-matrix technique [BGH03] relies on the approximation of the admissible blocks by analytic expansions of a given kernel function k. By replacing this approximation by ACA, $\mathcal{H}$-matrices become a purely algebraic approach based on matrix approximation.*

Relation to achieved results

Chapters 6 and 7, deal with the efficient reformulation and parallelization of $\mathcal{H}$-matrices on many-core hardware.

1.8 Overview of achieved results

In the following, the five papers and one preprint that form this cumulative habilitation are briefly discussed. The results are structured following the application scenario in which they are applied. In case a contribution is not single-authored, the results review is followed by a statement on the own contribution of the author.

1.8.1 Contributions in context of uncertainty quantification

Ensemble Kalman filters for reliability estimation in perfusion inference. In this work, which is single-authored and is available in the *International Journal for Uncertainty Quantification* under DOI `10.1615/Int.J.UncertaintyQuantification.2018024865`, the author solves the Bayesian inference problem discussed in Section 1.2.1 by the use of the Ensemble Kalman

Filter. While the Ensemble Kalman Filter technique, which uses Monte Carlo sampling, is known from the literature, the main contribution of this work is the modelling of the given imaging task as Bayesian inference problem. Previous work in the field solved the imaging task in a purely deterministic sense, e.g., by deconvolution and could therefore give no measure for the uncertainty present in the inferred predictions. The new approach allows to quantify these uncertainties giving additional reliability information to radiologists, which might improve diagnosis of, e.g., cancer and strokes in the future. This real-world application is a large-scale problem due to the excessive size of volumetric time-dependent imaging data.

Subspace correction methods in algebraic multi-level frames. This single-authored contribution is available in the journal *Linear Algebra and its Applications* under DOI `10.1016/j.laa.2015.09.026`. It introduces an *algebraic* multi-level frame construction. Algebraic multi-level frames are used to build a new class of iterative linear solvers for discretized elliptic problems on complex geometries. The new solvers are independent of these geometries and show optimal problem-size independent convergence rates. Similar solvers have been only available for geometrically constructed and thus *intrusive* multi-level frame approaches. The new algebraic construction was used as starting point of the next publication in uncertainty quantification.

On the algebraic construction of sparse multi-level approximations of elliptic tensor product problems. In this co-authored work, together with Helmut Harbrecht, which is published in Springer's *Journal of Scientific Computing* under DOI `10.1007/s10915-018-0807-6`, the second moment analysis discussed in Section 1.2.2 is solved by means of the newly introduced algebraically constructed sparse grid combination technique. The application is a large-scale problem that is a model for second moment analysis tasks in uncertainty quantification. While sparse grid combination technique approaches for this type of problem have been available beforehand, the new approach now works only with the information of a linear system arising in the discretization of one of the underlying domains. Thereby it becomes possible to apply the sparse grid combination technique to discretizations provided by arbitrary software packages. Moreover, this approach enables to apply the sparse grid combination technique without special handling of complex geometries.

Own contribution. The author of this thesis is the corresponding author the discussed article and would be the first author in a non-alphabetical ordering of the authors[1]. The idea and expertise for the application of the sparse grid combination technique to second moment analysis was contributed by Helmut Harbrecht. The author of this thesis contributed the knowledge on algebraic frames, did the numerical implementation and tests and wrote major parts of the manuscript.

1.8.2 Contributions in context of machine learning

Boosting quantum machine learning models with multi-level combination technique: Pople diagrams revisited. This co-authored work is published in the *Journal of Chemical Theory and Computation* under DOI `10.1021/acs.jctc.8b00832`. It introduces the sparse tensor

[1] Alphabetical ordering of the authors is the standard in mathematics.

product approximation, discussed in Section 1.6, to the field of machine learning based on kernel ridge regression. From a mathematical point of view, it combines approximation in reproducing kernel Hilbert spaces with the sparse grid combination technique / multi-index approximation. In the specific application, in which kernel ridge regression is called *Quantum Machine Learning*, the task is to train / approximate results of quantum chemistry calculations, cf. Section 1.2.3. The sparse grid combination technique is formally applied with respect to the number of molecules used in the machine learning, with respect to the applied quantum chemistry calculation model and with respect to the size of the so-called *basis set* used in each of the applied models. The resulting three-dimensional combination techique together with kernel ridge regression, allows to strongly outperform classical kernel ridge regression in the application field, when fixing a target training accuracy and comparing the number of most expensive quantum chemistry calculations that were necessary in the training to achieve this accuracy.

Own contribution. In this interdisciplinary collaboration, this thesis' author is the first author (in a non-alphabetical ordering). Further authors are (in this order) Bing Huang, Helmut Harbrecht and Anatole von Lilienfeld. While Anatole von Lilienfeld and Helmut Harbrecht combined the ideas of Quantum Machine Learning and the sparse grid combination technique, and Bing Huang provided the quantum chemical calculations, the author of this thesis developed and implemented the concrete realization of the proposed method. All numerical results with respect to "learning" were provided by the author. Moreover, a substantial contributions has been made with respect to the textual content of the article.

Algorithmic patterns for $\mathcal{H}$-matrices on many-core processors. In this single-authored work by the author, which is published in Springer's *Journal of Scientific Computing* under DOI `10.1007/s10915-018-0809-4`, the author introduces space filling curves as main spatial data structure to $\mathcal{H}$-matrices, cf. Section 1.7.3. Based on this reformulation of the method, the author is able to introduce the first fully many-core parallel implementation of the $\mathcal{H}$-matrix construction and $\mathcal{H}$-matrix-vector product based on the additional core components of a *many-core parallel tree traversal* and *batching of similar calculation tasks*. The main application for this matrix approximation technique in this article are the interpolation matrices $\boldsymbol{A}_{P_{k,X}}$ that show up in kernel-based interpolation, cf. Example 1.4, and that are used for training in the previous work on machine learning. Using the structuring by space filling curves and the other core components, strong performance improvements can be reported for the introduced many-core parallelization, comparing it to a multi-core parallel standard implementation on equally-priced hardware. This result will, in the long-run, lead to a much faster kernel ridge regression training / kernel-based approximation.

A scalable $\mathcal{H}$-matrix approach for the solution of boundary integral equations on multi-GPU clusters. This co-authored work, together with Helmut Harbrecht, is currently under review in the journal *Computers & Mathematics with Applications*. It is a follow-up work of the previously discussed $\mathcal{H}$-matrix work. Here, a second level of parallelization is introduced, leading to a scaling of the $\mathcal{H}$-matrix approach over several many-core processors. The specific focus of this work is further on the evaluation of kernel functions / matrix entries that are very computational expensive. This has been realized in context of boundary integral equations.

Note that the situation of an expensive kernel evaluation will, however, also be present in many machine learning applications, in which the parameter space is very high-dimensional.

Own contribution. The author of this thesis is the corresponding author of the discussed article and would be the first author in a non-alphabetical ordering of the authors. Knowledge on the boundary integral application and a sequential implementation were provided by Helmut Harbrecht. This thesis' author developed the parallelization beyond a single many-core hardware, parallelized the boundary integral equation code on many-core hardware, did all numerical tests and wrote a major part of the textual content.

References

[Beb00] M. Bebendorf. Approximation of boundary element matrices. *Numerische Mathematik*, 86(4):565–589, 2000.

[BG04] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:147–269, 2004.

[BG15] J. Ballani and L. Grasedyck. Hierarchical tensor approximation of output quantities of parameter-dependent PDEs. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):852–872, 2015.

[BGH03] S. Börm, L. Grasedyck, and W. Hackbusch. Introduction to hierarchical matrices with applications. *Engineering Analysis with Boundary Elements*, 27(5):405–422, 2003. Large scale problems using BEM.

[BMR82] A. Brandt, S. F. McCormick, and J. W. Ruge. Algebraic multigrid (AMG) for automatic multigrid solutions with applications to geodetic computations. Technical report, Inst. for Computational Studies, Fort Collins, CO, USA, 1982.

[BNT10] I. Babuka, F. Nobile, and R. Tempone. A stochastic collocation method for elliptic partial differential equations with random input data. *SIAM Review*, 52(2):317–355, 2010.

[BPX90] J. H. Bramble, J. E. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Mathematics of Computation*, 55(191):1–22, 1990.

[Caf98] R. Caflisch. Monte Carlo and quasi-Monte Carlo methods. *Acta Numerica*, 7:1–49, 1998.

[CD15] A. Cohen and R. DeVore. Approximation of high-dimensional parametric PDEs. *Acta Numerica*, 24:1–159, 2015.

[Dah97] W. Dahmen. Wavelet and multiscale methods for operator equations. *Acta numerica*, 6:55–228, 1997.

[DGLU15] Z. Dong, E. H. Georgoulis, J. Levesley, and F. Usta. Fast multilevel sparse Gaussian kernels for high-dimensional approximation and integration. *Preprint arXiv:1501.03296*, 2015.

[DS89] F. Delvos and W. Schempp. *Boolean methods in interpolation and approximation*. Pitman research notes in mathematics series. Longman Scientific & Technical, Harlow, Essex, England; Wiley, New York, USA, 1989.

[GH13] M. Griebel and H. Harbrecht. A note on the construction of L-fold sparse tensor product spaces. *Constructive Approximation*, 38(2):235–251, 2013.

[Gil15] M. B. Giles. Multilevel Monte Carlo methods. *Acta Numerica*, 24:259–328, 2015.

[Gri94] M. Griebel. Multilevel algorithms considered as iterative methods on semidefinite systems. *SIAM Journal on Scientific Computing*, 15(3):547–565, 1994.

[GSZ92] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In P. de Groen and R. Beauwens, editors, *Iterative Methods in Linear Algebra*, pages 263–281. IMACS, Elsevier, North Holland, 1992.

[GTY97] S. A. Goreinov, E. E. Tyrtyshnikov, and A. Y. Yeremin. Matrix-free iterative solution strategies for large dense linear systems. *Numerical Linear Algebra with Applications*, 4(4):273–294, 1997.

[Hac12] W. Hackbusch. *Tensor spaces and numerical tensor calculus*, volume 42 of *Springer series in computational mathematics*. Springer, Heidelberg, 2012.

[Hac15] W. Hackbusch. *Hierarchical Matrices: Algorithms and Analysis*. Springer, 1st edition, 2015.

[Ham09] J. Hamaekers. *Tensor Product Multiscale Many–Particle Spaces with Finite–Order Weights for the Electronic Schödinger Equation*. Phd thesis, Institute for Numerical Simulation, Universiy of Bonn, Germany, 2009.

[HANT16] A.-L. Haji-Ali, F. Nobile, and R. Tempone. Multi-index Monte Carlo: when sparsity meets sampling. *Numerische Mathematik*, 132(4):767–806, 2016.

[HANTT16] A.-L. Haji-Ali, F. Nobile, L. Tamellini, and R. Tempone. Multi-index stochastic collocation for random PDEs. *Computer Methods in Applied Mechanics and Engineering*, 306:95–122, 2016.

[Hei01] S. Heinrich. Multilevel Monte Carlo methods. In *Proceedings of the Third International Conference on Large-Scale Scientific Computing-Revised Papers*, LSSC '01, pages 58–67, London, UK, 2001. Springer, Berlin, Germany.

[HH94] G. Hämmerlin and K.-H. Hoffmann. *Numerische Mathematik*. Springer, Berlin, Germany, 4th edition, 1994.

[HPS12] H. Harbrecht, M. Peters, and R. Schneider. On the low-rank approximation by the pivoted Cholesky decomposition. *Applied Numerical Mathematics*, 62(4):428–440, 2012.

[HPS13] H. Harbrecht, M. Peters, and M. Siebenmorgen. Combination technique based k-th moment analysis of elliptic problems with random diffusion. *Journal of Computational Physics*, 252:128–141, 2013.

[HSS08] H. Harbrecht, R. Schneider, and C. Schwab. Multilevel frames for sparse tensor product spaces. *Numerische Mathematik*, 110(2):199–220, 2008.

[Lav67] M. Lavrent'ev. *Some Improperly Posed Problems of Mathematical Physics.* Springer tracts in natural philosophy. Springer, Berlin, Germany, 1967.

[Loè78] M. Loève. *Probability Theory II.* Graduate texts in mathematics. Springer, Berlin, Germany, 1978.

[Mar08] I. Markovsky. Structured low-rank approximation and its applications. *Automatica*, 44(4):891–909, 2008.

[NSW99] F. Narcowich, R. Schaback, and J. Ward. Multilevel interpolation and approximation. *Applied and Computational Harmonic Analysis*, 7(3):243–261, 1999.

[NTW08] F. Nobile, R. Tempone, and C. Webster. A sparse grid stochastic collocation method for partial differential equations with random input data. *SIAM J. Numer. Anal.*, 46(5):2309–2345, 2008.

[Ran17] R. Rannacher. *Numerik 0: Einführung in die Numerische Mathematik.* Heidelberg University Publishing, 2017.

[RC15] S. Reich and C. Cotter. *Probabilistic Forecasting and Bayesian Data Assimilation.* Cambridge University Press, 2015.

[RG18] A. Rüttgers and M. Griebel. Multiscale simulation of polymeric fluids using the sparse grid combination technique. *Applied Mathematics and Computation*, 319:425–443, 2018.

[RW17] C. Rieger and H. Wendland. Sampling inequalities for sparse grids. *Numerische Mathematik*, 136(2):439–466, 2017.

[SB05] J. Stoer and R. Bulirsch. *Numerische Mathematik 2.* Springer, Berlin, Germany, fifth edition, 2005.

[Smo63] S. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Soviet Mathematics, Doklady*, 4:240–243, 1963.

[ST03] C. Schwab and R.-A. Todor. Sparse finite elements for elliptic problems with stochastic loading. *Numerische Mathematik*, 95(4):707–734, 2003.

[Stu10] A. M. Stuart. Inverse problems: A Bayesian perspective. *Acta Numerica*, 19:451–559, 2010.

[VSL+15] K. Vu, J. C. Snyder, L. Li, M. Rupp, B. F. Chen, T. Khelif, K.-R. Müller, and K. Burke. Understanding kernel ridge regression: Common behaviors from simple functions to density functionals. *International Journal of Quantum Chemistry*, 115(16):1115–1128, 2015.

[Wen04] H. Wendland. *Scattered Data Approximation*. Cambridge University Press, 2004.

[WLB09] J. A. Witteveen, A. Loeven, and H. Bijl. An adaptive stochastic finite elements approach based on Newton-Cotes quadrature in simplex elements. *Computers & Fluids*, 38(6):1270–1288, 2009.

Part I

Contributions in context of uncertainty quantification

2 Ensemble Kalman filters for reliability estimation in perfusion inference

2.1 Introduction

Medical imaging by x-rays, *magnetic resonance imaging* (MRI) and *computed tomography* (CT) has considerably changed medical diagnosis throughout the last decades. Often, *contrast agents*, i.e. specific liquid chemicals, are injected into the patients blood circulation during the imaging process. This leads to *contrast–enhanced images* of higher contrast in some regions of the human body. In this work, we study inverse problems for a specific MRI or CT imaging task. That is, we aim at recovering a quantity of interest in medical imaging, which is derived by *dynamic* contrast–enhanced (DCE) imaging. In dynamic contrast–enhanced imaging, a time-dependent series of radiological images of a part of the patients body (e.g. the brain) is taken immediately after injecting a contrast agent into the patient's blood circulation. By observing the time-dependent concentration evolution of the contrast agent inside the patient's tissue, it is possible to recover information about the blood flow rates, i.e. the *perfusion*.

The outcome of the image acquisition process is a time-discrete series of tree-dimensional (space-discrete) concentration images $\boldsymbol{c}$ of a part of the patient body. The actual perfusion evaluation is a post-processing step, being preceded by image de-noising and motion compensation. Currently, blood perfusion is computed independently per discrete tissue volume element, i.e. *voxel*, thus spatial information is mostly neglected. Variants of the *indicator-dilution theory* [BGKW10, FKG$^+$11, Sou14] describe the concentration of a contrast agent in tissue at a given point in time as the result of a convolution in time of the (known) time-dependent arterial or blood circulation inflow concentration c_{art} with an unknown tissue-dependent kernel function $\boldsymbol{k}$. *Blood perfusion* is computed as a weighted maximum or point evaluation of the unknown kernel function.

Current state of the art methods aim at recovering the unknown time-dependent kernel function for given discrete measurements of the contrast agent in tissue. The kernel function is either modeled as a parametrized analytic function [Tof97, PRM$^+$06, FKG$^+$11] or discretized as a fully unknown function [BGKW10, ØWC$^+$96, FKG$^+$11]. Then, most approaches rely on a deterministic reconstruction of the kernel function, involving the solution of a deconvolution problem with regularization.

One drawback of the use of motion compensation, de-noising and deterministic deconvolution lies in the loss of information on the *quality* of a computed solution. That is, probabilistic information about the measurement accuracy and errors in space and time with their influence on the exactness on the computed quantity of interest are neglected or even lost.

In this work, we propose an approach to infer perfusion in the discussed application case while keeping the probabilistic information on the solution. Thereby, we overcome the discussed drawback of knowledge loss. To achieve this, we model the inference problem as a sequential data

assimilation problem: First, the unknown kernel function $\boldsymbol{k}$ is described as an unknown system state, for which a predictive time-discrete stochastic system state model is introduced. In this model, the kernel function is represented as a random variable. Then, a time-discrete stochastic observation model describes the relationship of the current approximation of $\boldsymbol{k}$ and the noisy measurements delivered by medical imaging. Finally, the well-known Ensemble Kalman Filter (EnKF) [Eve09, Stu10, ILS13, ESS15] is used to compute an ensemble–based approximation of the posterior probability density function (PDF) of $\boldsymbol{k}$ given the system state model and the (noisy) measurements. Based on this PDF, means, cumulative distribution functions, etc. can be computed. The whole sequential data assimilation methodology is applied to (noisy) artificial measurement data generated by a *Digital Perfusion Phantom* [RPV$^+$11, Pia12, Man], i.e. a forward model describing the mapping of perfusion information to medical images. Note that we stick to the use of a Digital Perfusion Phantom, instead of using exemplary patient data since, first, it allows to artificially create arbitrary amounts of radiological images and, second, a scanner- and patient-independent way to analyse perfusion estimation methods is highly desired in radiology. Certainly, applying the proposed methodology (together with radiologists) to real patient data is future work.

The data assimilation problem that we will model in Section 2.3 will stick to Gaussian random fields and a linear forward model. This is a strong simplification, allowing to analytically solve the data assimilation problem by the original *Kalman filter* [RC15], However, we use the EnKF, which is a generalization of this Kalman filter, usually being applied in a non-linear, non-Gaussian setting. For more details on the connection between EnKF and the Kalman filter and a convergence analysis in the linear Gaussian case see [MT18, Ton18], while EnKFs for inverse problems are discussed in [ILS13, SS17, SS18] and many extensions and alternatives for the EnKF are, e.g., developed in [And01, And03, TAB$^+$03, ZZH09, Rei13, RC13]. We decided to use the EnKF here, since we consider this work as a starting point for much more involved approaches for the prediction of perfusion. In fact, the rather simple linear forward model from the indicator dilution theory should be replaced by more complex or even PDE-based models, which will certainly no longer be linear. Moreover, we expect that a more involved evolution model, cf. Section 2.3.3, with non-Gaussian noise might become valid in real application cases. Therefore, we here already introduce the more involved EnKF framework, while considering other forward models and non-Gaussian noise as future work.

To the best of our knowledge, we consider the discussed work to be a new contribution to the field. Nevertheless, there has been previous work on the use of Ensemble Kalman Filters in the application scenario. In [NSL16], the authors concentrate on the introduction of a tissue model that includes space-dependent information. To achieve this, a blood flow model is combined with an EnKF. Preliminary results for this approach are given. In contrast, we focus here directly on the mathematical setting based on the indicator-dilution theory that is well-known and, therefore, well accepted by radiologist. Hence, our methodology is considered as an extension to the existing standard methodology introducing the opportunity to derive statistical information on the computed solution. In addition to the different objective compared to [NSL16], we also perform a large number of parameter studies and convergence tests, which are crucial to understand the properties of the method.

This article is organized as follows. In Section 2.2, we give a mathematical model for the radiological imaging and perfusion extraction mechanism. Section 2.3 outlines our numerical

approach based on sequential data assimilation using EnKF. Numerical results are given in Section 2.4 while Section 2.5 summarizes the discussed work.

2.2 Modeling radiological imaging and perfusion extraction

In the following, we start by giving an abstract model for the transport of contrast agent. Then, measurements by e.g. MRI are abstractly modeled. A concrete model for the contrast agent distribution is given by the indicator-dilution theory. Finally, our quantity of interest, i.e. blood perfusion, is introduced and the deterministic inference problem is summarized.

2.2.1 Abstract model for contrast agent transport

Shall $\mathcal{D}_{tiss} \subset \mathbb{R}^3$ be the tissue domain in the human body for which we want to derive information by dynamic contrast–enhanced imaging. We study contrast agent transport / concentrations in a time interval $[0, T] \subset \mathbb{R}$ with T being the final time. The inflow concentration of the contrast agent (at some arterial inlet) is a function $c_{art} : [0, T] \to \mathbb{R}_{\geqslant 0}$. The time-continuous contrast agent concentration in tissue can be modeled as a function $c : \mathcal{D}_{tiss} \times [0, T] \to \mathbb{R}_{\geqslant 0}$. Both are related to each other by an (unknown) operator $\mathcal{B}$, with

$$c(\cdot, t) = \mathcal{B}[c_{art}](t) \tag{2.1}$$

that models the function of the human body with respect to contrast agent transport.

2.2.2 Measuring contrast agent concentration in tissue

Appropriate measurement devices (CT, MRI, ...) usually have a cuboidal measurement domain. Therefore, we start by limiting $\mathcal{D}_{tiss}$ to $\mathcal{D}_{meas} = \times_{d=1}^{3}[0, a_d]$ with $\boldsymbol{a} = (a_1, a_2, a_3)^\top \in \mathbb{R}^3$ describing the size of the measurement domain. For simplicity, we assume the measurement domain and the area of interest to match exactly, i.e. $\mathcal{D}_{meas} = \mathcal{D}_{tiss}$, excluding cases in which some part of the measurement domain does not contain valid tissue. Moreover, $\mathcal{D}_{meas}$ is simplified as being stationary in time, i.e. the measurement device (or the patient) does not move or movements are considered as measurement error.

The finite spatial resolution $\boldsymbol{N}_{\mathcal{D}} \in \mathbb{N}^3$ of the measurement device leads to a decomposition of $\mathcal{D}_{meas}$ into $N_{voxel} = \prod_{d=1}^{3} N_{\mathcal{D}}^{(d)}$ volume elements or *voxels* of volume $V_{voxel} = \prod_{d=1}^{3} a_d / N_{\mathcal{D}}^{(d)}$ for which we obtain averaged (constant) measurements. We introduce a measurement operator Ψ that gives for a given exact contrast agent concentration c and a chosen point in time $t \in [0, T]$ a measurement vector $\boldsymbol{c}(t) \in \mathbb{R}_{\geqslant 0}^{N_{voxel}}$ as

$$\boldsymbol{c}(t) = \Psi[c](t) := \Theta[c](t) + \mathcal{E}[c](t)\,.$$

Here, Θ is a noise-free measurement-operator and is usually a volumetric average over each voxel being equivalent to a piece-wise constant approximation in space. $\mathcal{E}$ abstractly models a (potentially non-linear) additive error (noise, movements, technical problems, ...).

To reflect time-discrete measurements, we introduce N_{obs} ordered, pair-wise different discrete observation times $t_i^{obs} \in [0, T]$, $i \in \{1, \ldots, N_{obs}\}$, at which measurements or observations are

done, giving the observation matrix $\boldsymbol{C} := (c_{ji})_{j=1,\dots,N_{voxel},i=1,\dots,N_{obs}}$ composed of observation vectors $\boldsymbol{c}_i$ as $\boldsymbol{C} = (\boldsymbol{c}_1|\dots|\boldsymbol{c}_{N_{obs}})$ with

$$\boldsymbol{c}_i = \Psi[c](t_i^{obs}) := \Theta[c](t_i^{obs}) + \mathcal{E}[c](t_i^{obs})\,. \tag{2.2}$$

2.2.3 Contrast agent transport model following the indicator-dilution theory

The indicator-dilution theory (IDT) [BGKW10] provides a model for the time evolution of the contrast agent concentration in a reference voxel $\mathcal{D}_{voxel} \subset \mathcal{D}_{tiss}$ with volume V_{voxel}, given the arterial inflow c_{art}. While, in this standard model, the contrast agent's concentrations are assumed to be constant in each voxel, we first want to formulate the IDT as a space-continuous model and then move over to a discrete description as consequence of a measurement process. Our continuous version of the indicator-dilution theory–based transport model replaces $\mathcal{B}$ in eq. (2.1) with the model operator $\mathcal{B}_{IDT}$ given via

$$c(\boldsymbol{x},t) = \mathcal{B}_{IDT}[c_{art},k](t) := \int_0^T c_{art}(\tau)k(\boldsymbol{x},t-\tau)d\tau\,, \quad (\boldsymbol{x},t) \in \mathcal{D}_{tiss} \times [0,T]\,. \tag{2.3}$$

Kernel $k : \mathcal{D}_{tiss} \times [0,T] \to \mathbb{R}$ fully characterizes the properties of the tissue at point $\boldsymbol{x}$. In order to have an well-defined integrand, we assume $k(\cdot,t) = 0$ for $t < 0$. Note that the model operator $\mathcal{B}_{IDT}$ is actually independent of the spatial position.

We now apply the measurement operator Ψ to eq. (2.3) obtaining

$$\begin{aligned}\Psi[c](t) &= \Theta\left[\mathcal{B}_{IDT}[c_{art},k]\right](t) + \mathcal{E}\left[\mathcal{B}_{IDT}[c_{art},k]\right](t)\\ &= \int_0^T c_{art}(\tau)\boldsymbol{k}(t-\tau)d\tau + \mathcal{E}\left[\mathcal{B}_{IDT}[c_{art},k]\right](t)\,,\end{aligned}$$

where $\boldsymbol{k} = (k_1,\dots,k_{N_{voxel}})^\top$ is a vector of univariate kernel functions $k_j : [0,T] \to \mathbb{R}$. Since we are interested in time-discrete observations, we limit our discussion to observation times t_i^{obs} yielding

$$\boldsymbol{c}_i = \Psi[c](t_i^{obs}) = \int_0^T c_{art}(\tau)\boldsymbol{k}(t_i^{obs}-\tau)d\tau + \boldsymbol{e}_i(c_{art},k)\,,$$

with the abbreviation $\boldsymbol{e}_i(c_{art},k) := \mathcal{E}\left[\mathcal{B}_{IDT}[c_{art},k]\right](t_i^{obs})$. For a single voxel $j \in \{1,\dots,N_{voxel}\}$, we obtain

$$c_{j,i}^{obs} = \int_0^T c_{art}(\tau)k_j(t_i^{obs}-\tau)d\tau + e_{j,i}(c_{art},k)\,.$$

In case of $e_{j,i}(c_{art},k) = 0$, this boils down to the classical indicator-dilution-theory model given on a reference voxel j. Obviously, this model is independent of the spatial position of the voxel j. The classical theory further introduces a mean density $\rho_j \in \mathbb{R}_{\geqslant 0}$ in a voxel j, which becomes of interest in the following subsection.

2.2.4 Perfusion

The inference task discussed in this article is to compute a time-stationary perfusion (blood flow) information $\boldsymbol{p} \in \mathbb{R}^{N_{voxel}}$ given the (assumed to be exactly known) inflow concentration c_{art} and the observation matrix $\boldsymbol{C}$. Formally, the blood perfusion in a given voxel j can be evaluated as quantity of interest of the computed response function k_j as

$$p_j := p(k_j) := \frac{1}{\rho_j} k_j(0)\,.$$

From a mathematical point of view, this quantity has nice properties, since it is just a point evaluation of the response function. In practice [BGKW10], perfusion is however often evaluated as

$$\tilde{p}_j := \tilde{p}(k_j) := \frac{1}{\rho_j} \max_{t\in[0,T]} k_j(t)\,.$$

For simplicity and since we use just artificial input data, we stick to the first version of this quantity of interest.

2.2.5 Deterministic inference problem

To summarize this section, we formulate the deterministic problem that we aim to solve: For given measurement time $T \in \mathbb{R}$, arterial inflow $c_{art} : [0,T] \to \mathbb{R}_{\geqslant 0}$, measurement/observation times $t_1^{obs} < t_2^{obs} < \ldots < t_{N_{obs}}^{obs}$ and observation matrix $\boldsymbol{C}$ or vectors $\boldsymbol{c}_i \in \mathbb{R}^{N_{voxel}}$, $i \in \{1,\ldots,N_{obs}\}$, we aim at computing a vector $\boldsymbol{k} = (k_1,\ldots,k_{N_{voxel}})^T$ of kernel functions $k_j : [0,T] \to \mathbb{R}$ and the derived quantity of interest $\boldsymbol{p} = (p_1,\ldots,p_{N_{voxel}})^\top$ with $p_j = k_j(0)/\rho_j$ such that

$$c_{j,i} \approx \int_0^T c_{art}(\tau) k_j(t_i^{obs} - \tau) d\tau + e_{j,i}(c_{art}, k)\,, \qquad j \in \{1,\ldots,N_{voxel}\},\ i \in \{1,\ldots,N_{obs}\}\,. \quad (2.4)$$

Clearly, this problem is underdetermined with the given requirements. Furthermore, we have not specified the nature of the error term, yet. This is why we used the notion "$\approx$". A much clearer idea of the concept of a *solution* to this problem is given in the next section, where we reformulate the problem as Bayesian sequential data assimilation problem.

2.3 Numerical approach by sequential data assimilation

In this section, we first introduce a discretization for the model discussed in the last section. This is necessary, since we will use its discretized version in context of sequential data assimilation, afterwards. An approximation to the solution of the assimilation problem is derived by the Ensemble Kalman Filter that is briefly introduced as final part of this section.

2.3.1 Discretized observation model

We start by discretizing eq. (2.4) for fixed $i \in \{1, \dots, N_{obs}\}$ and fixed $j \in \{1, \dots, N_{voxel}\}$. Numerical quadrature using a rectangular rule gives

$$c_{j,i} \approx \Delta\tau \sum_{q=0}^{N_q-1} c_{art}(\tau_q) k_j(t_i^{obs} - \tau_q) + e_{j,i}(c_{art}, k)\,,$$

with N_q equidistant abscissas $\tau_q := q \cdot \Delta\tau$ and $\Delta\tau := \frac{T}{N_q}$. In the original problem setting, the observation times t_i^{obs} can be chosen arbitrarily. However, we here introduce a simplification, in which we assume the observation times to be given for a fixed time step size Δt_{obs}. Moreover, this time step size shall be a multiple of $\Delta\tau$, i.e.

$$t_i^{obs} := i\,\Delta t_{obs}\,, \quad \Delta t_{obs} := s \cdot \Delta\tau\,, \quad s \in \mathbb{N}\,.$$

Thereby, we obtain

$$\begin{aligned} c_{j,i} &\approx \Delta\tau \sum_{q=0}^{N_q-1} c_{art}(\tau_q) k_j(t_i^{obs} - \tau_q) + e_{j,i}(c_{art}, k) \\ &= \Delta\tau \sum_{q=0}^{N_q-1} c_{art}(q\Delta\tau) k_j\left((i\,s - q)\Delta\tau\right) + e_{j,i}(c_{art}, k)\,. \end{aligned}$$

Since we now only need c_{art} and k_j being evaluated at multiples of $\Delta\tau$, we can replace them by vector $\boldsymbol{c}_{art} = \left(c_{art,0}, \dots, c_{art,N_q-1}\right)^\top$ such that $c_{art,q} := c_{art}(q\Delta\tau)$ and matrix $\boldsymbol{K} \in \mathbb{R}^{N_q \times N_{voxel}}$ with $\boldsymbol{K} := (k_{q,j})_{q,j}$ such that $k_{q,j} := k_j(q\Delta\tau)$, yielding

$$c_{j,i} \approx \Delta\tau \sum_{q=0}^{N_q-1} c_{art,q}\, k_{(i\,s-q),j} + e_{j,i}(c_{art}, k)\,.$$

With the extension of $k_j(t) = 0$ for $t < 0$ and some index substitutions, we can finally find (for each j, i) a (degenerated) matrix $\boldsymbol{H}_{j,i} \in \mathbb{R}^{1 \times N_q}$ such that

$$c_{j,i} \approx \boldsymbol{H}_{j,i}\boldsymbol{k}_j + e_{j,i}(c_{art}, k)\,, \tag{2.5}$$

where the $\boldsymbol{k}_j \in \mathbb{R}^{N_q}$ are the column vectors of matrix $\boldsymbol{K}$, i.e. $\boldsymbol{K} = (\boldsymbol{k}_0 | \dots | \boldsymbol{k}_{N_q-1})$.

Following the nomenclature of [RC15], we next reformulate the deterministic inference problem from Section 2.2.5 as a sequential data assimilation problem. To this end, we first translate the involved quantities into random variables as in a Bayesian inference problem. Thereafter, we introduce the basic concepts of sequential data assimilation.

Since the problem decouples for all voxels $j \in \{1, \dots N_{voxel}\}$, we keep j fixed for the rest of this section.

2.3.2 Probabilistic view of inference

Let be $(\Omega, \mathcal{F}, \mathbb{P})$ a probability space. In Bayesian inference we want to gain information on a *system state variable* for given observation(s). In our context, the state variable is the time-continuous kernel function k_j. However, for simplicity and since we deal with discrete data anyway, we infer the discrete $\boldsymbol{k}_j \in \mathbb{R}^{N_q}$ from eq. (2.5), instead. Therefore, we introduce a new random variable

$$\mathsf{k}_j : \Omega \to \mathbb{R}^{N_q}\,, \tag{2.6}$$

replacing the time-discrete deterministic solution vector $\boldsymbol{k}_j$.[1] Moreover, we introduce a random variable $\mathsf{e}_{j,i}(c_{art}) : \Omega \to \mathbb{R}$, replacing the error term used before. Note that we assume k_j and $\mathsf{e}_{j,i}$ to be independent random variables. This is a rather strong simplification, since we initially modeled $e_{j,i}(c_{art}, k)$ to be a potentially non-linear error in the observation data, which itself is given in the indicator-dilution-theory by the arterial inflow c_{art} and the tissue properties modeled by kernel k. That is, we – at this point – decouple the error in the observation from the specific patient tissue. This decoupling is reflected by the new notation $\mathsf{e}_{j,i}(c_{art})$. Finally, we also consider each observation $c_{j,i}$ as random variable $\mathsf{c}_{j,i} : \Omega \to \mathbb{R}$, which is usually called *observed variable*. Using eq. (2.5), $\mathsf{c}_{j,i}$ is defined as

$$\mathsf{c}_{j,i}(\omega) := \boldsymbol{H}_{j,i}\mathsf{k}_j(\omega) + \mathsf{e}_{j,i}(c_{art})(\omega), \qquad \forall \omega \in \Omega\,. \tag{2.7}$$

We will call matrix $\boldsymbol{H}_{j,i}$ *(linear) forward map*. The aim of inference is to find a reference *trajectory* $\boldsymbol{k}_j^{ref}$, being a realization of k_j such that the (measured) observations fit to the observed variable.

2.3.3 Sequential data assimilation

Sequential data assimilation relies on an *evolution model* and a *forward model* to obtain $\boldsymbol{k}_j^{ref}$. The models run on different time scales. The evolution model is a stochastic difference equation implying a certain predicted evolution of the system state variable over many small time steps. The forward model defines a relationship between the reference trajectory (which is to be found) and the observed data at the observation times.

Evolution model

In context of sequential data assimilation for dynamic processes, it is usually assumed that the coupling between the measured observations and the system state variable is time-local. That is, a new observation at time t_{obs} only affects the system state variable for times $t \geqslant t_{obs}$. In our application, this is different, since the forward model, i.e. the indicator-dilution theory, is a non-local operator in time. Therefore, we need an evolution model that allows to do global updates to the system state variable k_j. The probably most simplistic approach to model this type of global updates is given by the evolution model

$$\mathsf{k}_j^{(l+1)} = \mathsf{k}_j^{(l)} + \sqrt{\Delta\tau}\,\mathsf{n}^{(l)}\,, \quad l \in \{0, \ldots, N_q - 1\}\,, \tag{2.8}$$

[1] We use sans serif letters to indicate that a given quantity is a random variable.

Here $\left(\mathbf{k}_j^{(0)}, \ldots, \mathbf{k}_j^{(N_q-1)}\right)$, is a sequence of random variables of the type given in eq. (2.6) for time steps $l\,\Delta\tau$. We assume $\mathbf{k}_j^{(0)} \sim \mathcal{N}(\mathbf{0}, {\sigma_0}^2\boldsymbol{\Sigma}_\mathbf{n})$, corresponding to a zero initial guess for the kernel function with Gaussian noise with a covariance matrix $\boldsymbol{\Sigma}_\mathbf{n}$. $\sigma_0 \in \mathbb{R}$ is a scaling coefficient. The $\mathbf{n}^{(l)}$s are a sequence of independent identically distributed random variables with $\mathbf{n}^{(n)} : \Omega \to \mathbb{R}^{N_q}$ drawn as $\mathbf{n}^{(l)} \sim \mathcal{N}(0, \boldsymbol{\Sigma}_\mathbf{n})$. $\boldsymbol{\Sigma}_\mathbf{n} \in \mathbb{R}^{N_q \times N_q}$ will be chosen using a Gaussian covariance kernel such that $\boldsymbol{\Sigma}_\mathbf{n} := \left(\sigma_{l,l'}\right)_{l,l'=0}^{N_q-1}$ with $\sigma_{l,l'} := \alpha e^{-\frac{\|\tau_l - \tau_{l'}\|_2^2}{2\ell^2}}$ and a parametrization in the scale $\alpha \in \mathbb{R}$ and the correlation length $\ell \in \mathbb{R}$.

Analyzing this evolution model, we can state that it can be understood as Euler-Maruyama-based discretization of the system of stochastic ordinary differential equations

$$d\mathbf{k}_j = d\boldsymbol{W}_\boldsymbol{t}\,, \tag{2.9}$$

where $\boldsymbol{W}_\boldsymbol{t}$ is a vector of correlated univariate Wiener processes. Moreover, we observe that this evolution model, in contrast to the standard setting of dynamical processes, now only takes the role of coupling the time-discrete values in $\mathbf{k}$. This coupling is imposed by the covariance of the noise term. In fact, as we will see in Section 2.4.5, the *correlation length* in the Gaussian covariance kernel will have a regularizing influence on the inferred solution. Note that the choice of Gaussian noise might lead to a locally negative kernel $\boldsymbol{K}$, while this kernel is supposed to be positive. The choice of a better noise distribution is future work.

A rather natural question in context of the proposed application is, whether it would preferable to apply an EnKF-based approach for direct inversion, cf. [ILS13, SS17, SS18] to the measurement matrix $\boldsymbol{C}$, avoiding *sequential* data assimilation. In fact, we prefer sequential data assimilation, since it allows to treat the given inference problem as a time-dependent problem. In the real application case of DCE imaging, one objective of researchers is to find means to effectively control the image capturing process, in terms of a feedback loop. In that context, it is important to be able to continuously monitor the achieved approximation of the perfusion information *during* the image capturing process. Based on that monitoring, one might be able to select the next observation time or the required quality for the next observation. It is more natural to achieve this control approach by data assimilation than by direct inversion.

Forward model

We choose eq. (2.7) as our forward model, i.e. we get the forward model with respect to the reference trajectory $\boldsymbol{k}_j^{ref}$

$$\mathsf{c}_{j,i} = \boldsymbol{H}_{j,i} k_{j,i\,s}^{ref} + \mathsf{e}_{j,i}\,, \quad i \in \{1, \ldots, N_{obs}\}\,. \tag{2.10}$$

The $\mathsf{e}_{j,i}$ are sequences of i.i.d. random variables for growing observation time index i following $\mathsf{e}_{j,i} \sim \mathcal{N}(0, \sigma_\mathsf{e})$, for all $i \in \{1, \ldots, N_{obs}\}$, with $\sigma_\mathsf{e} \in \mathbb{R}$ the observation error variance. Note that this choice of the distribution of $\mathsf{e}_{j,i}$ is a further simplification over the simplification that has been made in Section 2.3.2. There, we decoupled the observation error from the kernel function k describing the tissue properties. Here, we further decouple the observation error from the arterial inflow and make it a purely data-independent error that is furthermore only modeled

as normally distributed. It is very clear, that this is a very strong simplification. Finding a much better, maybe imaging device dependent, error is future work.

Due to $\Delta t_{obs} = s\Delta\tau$, $k^{ref}_{j,i\,s}$ is the unknown reference trajectory evaluated at observation time t^{obs}_i.

Assimilation task

To be concise, we here only briefly summarize the general idea of the actual assimilation task with notation from [RC15]. Further details can be found e.g. in [RC15].

Let $\pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j)$ be the probability density function of the random variable $\mathbf{k}_j^{(i\,s)}$ at time t_i^{obs} for $i \in \{1, \dots, N_{obs}\}$. Then, sequential data assimilation computes posterior PDFs

$$\pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j | c_{j,1:i}) \,, \quad i \in \{1, \dots, N_{obs}\} \,,$$

i.e. probability density functions of the random variables $\mathbf{k}_j^{(i\,s)}$ with an instance $\boldsymbol{k}_j$ conditioned to the observations $c_{j,1}, \dots, c_{j,i}$ that are instances of $\mathsf{c}_{j,1}, \dots, \mathsf{c}_{j,i}$. This is done using an iterative approach. It is started with $\pi_{\mathbf{k}_j^{(0)}}(\boldsymbol{k}_j | c_{j,1:0})$ being the PDF of $\mathbf{k}_j^{(0)}$. Then, for a given PDF $\pi_{\mathbf{k}_j^{((i-1)\,s)}}(\boldsymbol{k}_j | c_{j,1:i-1})$, it iteratively

1. computes the density $\pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j | c_{j,1:i-1})$ and thereby solves a prediction problem for the given evolution model eq. (2.8),
2. applies Bayes theorem

$$\pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j | c_{j,1:i}) = \frac{\pi_{\mathsf{c}_{j,i}}(c_{j,i} | \boldsymbol{k}_j)\, \pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j | c_{j,1:i-1})}{\int_{\mathbb{R}^{N_q}} \pi_{\mathsf{c}_{j,i}}(c_{j,i} | \boldsymbol{k}_j)\, \pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j | c_{j,1:i-1}) d\boldsymbol{k}_j}$$

   in an update step to compute $\pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j | c_{j,1:i})$.

In other words, the idea is to start from knowledge (encoded in $\pi_{\mathbf{k}_j^{((i-1)\,s)}}(\boldsymbol{k}_j | c_{j,1:i-1})$) at an observation time step t^{obs}_{i-1}. Then, knowledge for a new observation time step is forecasted / predicted using only the evolution model eq. (2.8). This forecast is finally corrected using the information given by observation $c_{j,i}$. The unknown reference trajectory is ultimately given as mean of the marginal PDF $\pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j | c_{j,1:N_{obs}})$.

2.3.4 Ensemble Kalman Filter

The EnKF is a Monte-Carlo–type implementation of the above discussed iterative data assimilation task. Instead of explicitly computing the posterior PDFs $\pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j | c_{j,1:i-1})$ and $\pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j | c_{j,1:i})$, the EnKF constructs an *ensemble of realizations* of random variables representing these PDFs in an empirical sense. In that context, *forecast* and *analysis ensembles* are distinguished. As we will see, the computation of the forecast ensemble corresponds to

approximating $\pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j|c_{j,1:i-1,j})$, while the computation of the analysis ensemble corresponds to the approximation of $\pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j|c_{j,1:i,j})$.

Shall N_e be the size of the ensembles. Then, the EnKF algorithm starts by drawing N_e samples $\boldsymbol{k}_j^{(0),1},\dots,\boldsymbol{k}_j^{(0),N_e}$ of the (initial) system state according to the PDF of $\mathbf{k}_j^{(0)}$. The algorithm consists of two main steps which are iteratively done for $i \in \{1,\dots,N_{obs}\}$.

Forecast step

In the forecast step, the ensemble is propagated over s steps of the evolution model in eq. (2.8) to reach the next observation time step $t_i^{obs} = i\,s\,\Delta\tau$. To achieve this, realizations $\boldsymbol{n}^{(l),m} \in \mathbb{R}^{N_q}$ for $m \in \{1,\dots,N_e\}$ are drawn i.i.d. from $\mathbf{n}^{(n)}$ in each of the s steps. Then the propagation equation reads for $n = 1,\dots,s$ as

$$\boldsymbol{k}_j^{(i(s-1)+l),m} = \boldsymbol{k}_j^{(i(s-1)+(l-1)),m} + \sqrt{\Delta\tau}\,\boldsymbol{n}^{(l),m}, \quad m \in \{1,\dots,N_e\}\,.$$

The newly generated ensemble is the *forecast ensemble* $\left(\boldsymbol{k}_j^{f,m}\right)_{m=1}^{N_e}$ with $\boldsymbol{k}_j^{f,m} := \boldsymbol{k}_{\boldsymbol{j}}^{i\,s,m}$. We further compute the empirical forecast mean

$$\overline{\boldsymbol{k}_j^f} := \frac{1}{N_e}\sum_{m=1}^{N_e} \boldsymbol{k}_j^{f,m} \in \mathbb{R}^{N_q} \tag{2.11}$$

and the empirical forecast covariance (matrix)

$$\boldsymbol{\Sigma}_{\boldsymbol{k}_j}^f := \frac{1}{N_e-1}\sum_{m=1}^{N_e}\left(\boldsymbol{k}_j^{f,m} - \overline{\boldsymbol{k}_j^f}\right)\left(\boldsymbol{k}_j^{f,m} - \overline{\boldsymbol{k}_j^f}\right)^\top \in \mathbb{R}^{N_q\times N_q}\,. \tag{2.12}$$

Analysis step

In the analysis step, the Kalman filter [K$^+$60, RC15] is applied to the forecast ensemble to compute an analysis ensemble $\left(\boldsymbol{k}_j^{a,m}\right)_{m=1}^{N_e}$ representing the PDF $\pi_{\mathbf{k}_j^{(i\,s)}}(\boldsymbol{k}_j|c_{j,1:i})$, which is conditioned to the new observation $c_{j,i}$. As part of the Kalman filter, the forward model eq. (2.10) with $k_{j,i\,s}^{ref}$ being replaced by $\mathbf{k}_j^{(i\,s)}$ is evaluated. Here, we use a linear forward map $\boldsymbol{H}_{j,i}$. Moreover all involved random variables are Gaussian. Therefore, it can be shown that the analysis ensemble follows a Gaussian distribution, too and thus it can be fully characterized by the empirical analysis mean $\overline{\boldsymbol{k}_j^a}$ and the empirical analysis covariance $\boldsymbol{\Sigma}_{\boldsymbol{k}_j}^a$.

Based on this observation, the core idea of the Kalman filter is to compute the empirical analysis mean as minimization problem

$$\overline{\boldsymbol{k}_j^a} = \underset{\boldsymbol{k}_j\in\mathbb{R}^{N_q}}{\operatorname{argmin}}\,\frac{1}{2}\left(\left\|\boldsymbol{k}_j - \overline{\boldsymbol{k}_j^f}\right\|^2_{\left(\boldsymbol{\Sigma}_{\boldsymbol{k}_j}^f\right)^{-1}} + \left\|\boldsymbol{H}_{j,i}\boldsymbol{k}_j - c_{j,i}\right\|^2_{\sigma_{\mathrm{e}}^{-1}}\right)\,.$$

Given the linearity of $\boldsymbol{H}_{j,i}$, the minimum can be exactly computed as

$$\overline{\boldsymbol{k}_j^a} = \overline{\boldsymbol{k}_j^f} - \boldsymbol{U}_{j,i}(\boldsymbol{H}_{j,i}\boldsymbol{k}_j - c_{j,i})\,,$$

where $U_{i,j}$ is the Kalman (update) matrix

$$\boldsymbol{U}_{j,i} = \boldsymbol{\Sigma}_{\boldsymbol{k}_j^f}\boldsymbol{H}_{j,i}^\top(\boldsymbol{H}_{j,i}\boldsymbol{\Sigma}_{\boldsymbol{k}_j}^f\boldsymbol{H}_{j,i}^\top + \sigma_{\mathsf{e}})^{-1}\,.$$

Instead of explicitly computing the empirical analysis mean and covariance (the latter by an analogous update idea), the analysis part of the Ensemble Kalman Filter (with perturbed observations) [RC15, Chapter 7] directly updates the forecast ensemble by

$$\boldsymbol{k}_j^{a,m} = \boldsymbol{k}_j^{f,m} - \boldsymbol{U}_{j,i}(\boldsymbol{H}_{j,i}\boldsymbol{k}_j^{f,m} + e_{j,i,m} - c_{j,i})\,, \quad m \in \{1,\dots,N_e\}\,,$$

where $\{e_{j,i,m}\}_{m=1}^{N_e}$ are realizations of $\mathsf{e}_{j,i}$. If required, empirical versions of the analysis mean and analysis covariance can be computed analogously to eq. (2.11) and eq. (2.12). Finally, the next forecast step is initialized with $\boldsymbol{k}_j^{(i\,s),m} = \boldsymbol{k}_j^{a,m}$, that is, the analysis ensemble replaces the system state for t_i^{obs}.

Result

For $i = N_{obs}$ the algorithm terminates with an analysis ensemble, representing the posterior PDF $\pi_{\mathbf{k}_j^{(N_{obs}\cdot s)}}(\boldsymbol{k}_j|c_{j,1:N_{obs}})$. The reference trajectory is extracted as empirical mean $\overline{\boldsymbol{k}_j} := \frac{1}{N_e}\sum_{m=1}^{N_e}\boldsymbol{k}_j^{(N_{obs}\,s),m}$. The (mean) perfusion $\overline{p_j}$ can be derived as $\overline{p_j} = \frac{1}{\rho_j}\overline{\boldsymbol{k}_j}|_{t=0}$. Empirical covariances are extracted as discussed before. Moreover, in case cumulative distribution functions or other probabilistic quantities shall be extracted, a kernel-density estimator (such as `ksdensity` in Matlab) is applied to the generated ensemble.

2.4 Numerical results

In this section, we demonstrate the beforehand introduced numerical method for artificial test data. To this end, we first introduce the source of this test data, which is a *Digital Perfusion Phantom*. Then, we study the numerical properties of our method in terms of convergence, parameter dependence and input dependence in a single-voxel scenario. Finally we solve the perfusion inference problem for a slice of a full (artificial) DCE imaging brain data set.

2.4.1 Digital Perfusion Phantom

Digital Perfusion Phantoms (DPP) [RPV$^+$11, Pia12, Man] allow to artificially generate DCE image data for perfusion analysis. Thereby new algorithms can be tested on such data without the additional constraints of true patient data. Perfusion Phantoms basically solve the forward problem, which involves to transform perfusion information into contrast agent concentrations. In our work, we use the *Digital Brain Perfusion Phantom* package [Man], which is a Matlab implementation of the model introduced in [RPV$^+$11]. The software provides a radiological

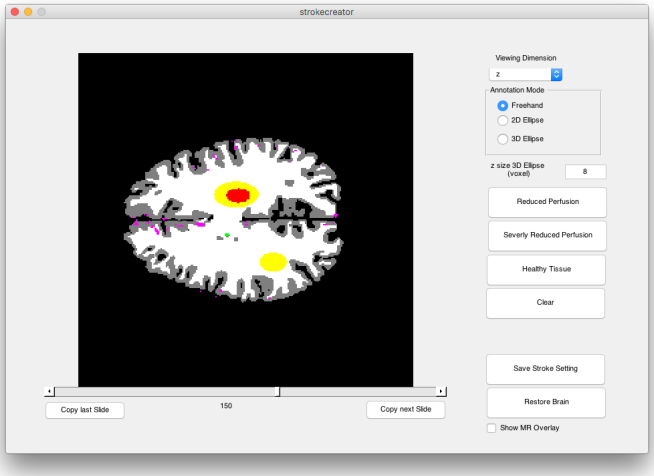


Figure 2.1: The source of our artificial measurements is the Digital Brain Perfusion Phantom package [Man]. A Matlab implementation of this work is available. It allows to mark brain regions with reduced and severely reduced perfusion, here shown with the colors yellow and red. Given this data, artificial DCE imaging data is are created.

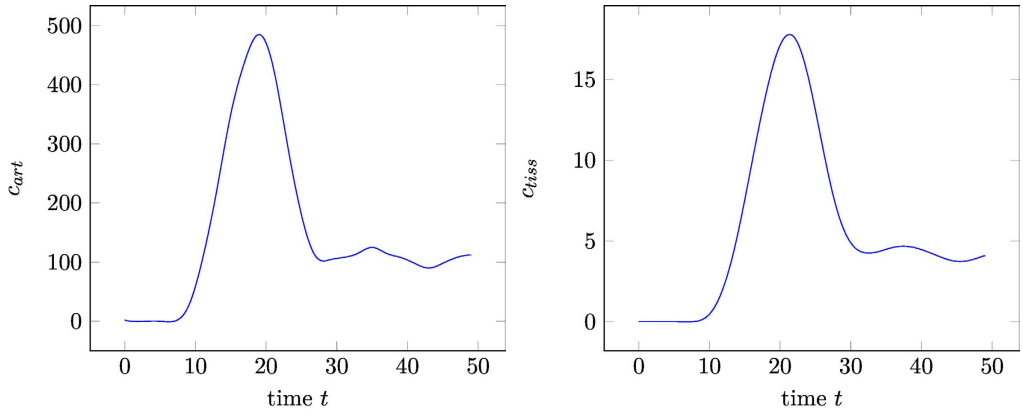
image of a brain. A user interface, see Figure 2.1, allows to mark regions of reduced and strongly reduced perfusion. It is possible to control the observation snapshot time step size (i.e. Δt_{obs}) of the artificial radiological imaging process. The measurement time is $T = 49$. The resolution of the artificially generated data is $\boldsymbol{N}_{\mathcal{D}} = (256, 256, 256)$. The arterial input function is provided as discrete evaluations $c_{art}(t_i^{input})$ with $t_i^{input} = 2\,i$. The Perfusion Phantom package uses a piecewise cubic spline interpolant through this data as exact c_{art}, see Figure 2.2(a). During the artificial imaging process, each snapshot (i.e. $\boldsymbol{c}_i$) is written in a separate file. A *baseline* for the radiological images is written, too. It contains the measurement data without contrast agent concentrations. In our examples, we always subtract this baseline data from the artificial measurements to obtain just the necessary concentration information.

We perform a major part of our numerical tests on a single reference voxel which has been chosen arbitrarily as $(100, 130, 150)$. The observation data for that single voxel is stored with a time step size of $\Delta t_{obs} = 0.25$. This data is interpolated by a piecewise cubic spline to obtain measurement data at arbitrary points in time for our initial tests, cf. Figure 2.2(b). Towards the end of this section, results for a full slice $(\cdot, \cdot, 150)$ of the full data set are discussed.

We start by showing a series of numerical results obtained for given artificial input without noise. These results will give an insight into the choice of the different parameters of the method and into the convergence properties of the method. Noisy data is discussed afterwards.

2.4.2 Data assimilation process

Let us first have a look at the evolution of the analysis ensemble during the sequential data assimilation process. We have chosen an observation time step size of $\Delta t_{obs} = 0.25$, a quadra-



(a) arterial input function (b) concentration measurement in single voxel

Figure 2.2: We use idealized concentration functions for one tissue voxel in order to test the implemented numerical method.

ture step size of $\Delta\tau = 0.0625$ (i.e. $s = 4$) and an ensemble size of $N_e = 5000$. For a meaningful definition of the (co-)variances, we have to account for the scales of the involved quantities. By experiments, we found out that the kernel function k_j has a magnitude of about 10^{-3}. Therefore, the scaling α of the covariance matrix $\mathbf{\Sigma_n}$ should be relative to a *standard deviation* of 10^{-3}. With this in mind, we set $\alpha = (10^{-3})^2\, 0.001$. This corresponds to a *relative variance* of 0.001. Note that it would be highly desirable to perform a coupled inference of the kernel function k_j *and* the scaling α. This is considered future work. The correlation length is set to $\ell = 2$. For the covariance of the initial state $\mathbf{k}_j^{(0)}$, we impose an additional scaling of $\sigma_0 = 100$, accounting for a much larger uncertainty in the initial state. The observation error variance also needs a problem-adapted scaling. Since the concentration measurements are in the range of 10, we shift the (co-)variance by a standard deviation of 10. Using a relative variance of 0.0001, we obtain $\sigma_{\mathbf{e}} = 10^2\, 0.0001$.

In Figure 2.3, we show the evolution of the empirical mean $\overline{\boldsymbol{k}_j^a}$, i.e. the prediction for the unknown kernel function, for different observation times during the operation of the EnKF. Note that a scaled evolution of $\overline{\boldsymbol{k}_j^a}$ at $t = 0$ corresponds to the (scaled) unknown perfusion $\overline{\boldsymbol{p}}$. Therefore, discussing numerical results for $\overline{\boldsymbol{k}_j^a}$ is equivalent to discussing results for $\overline{\boldsymbol{p}}$. The major information gain for the predicted result is in time interval $[10, 20]$. This is the time interval in which the concentration at the arterial inlet grows. Afterwards, the data assimilation process only gains very little more information and converges towards the final result.

2.4.3 Convergence in the ensemble size

Next, we discuss the convergence of the empirical mean of $\boldsymbol{k}_j$ with respect to the ensemble size N_e. In the following, we will always concentrate on the last analysis ensemble obtained after assimilating the observation for $t_{obs} = 49$. To shorten notation, we skip additional indices, indicating this and call the empirical mean of this analysis ensemble $\overline{\boldsymbol{k}_j}$.

Our convergence study with respect to the ensemble size uses the same parameters as in the

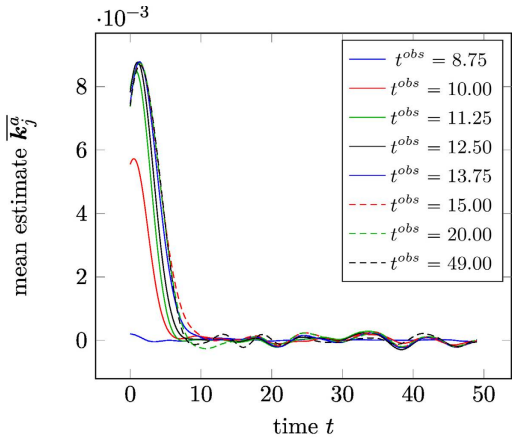


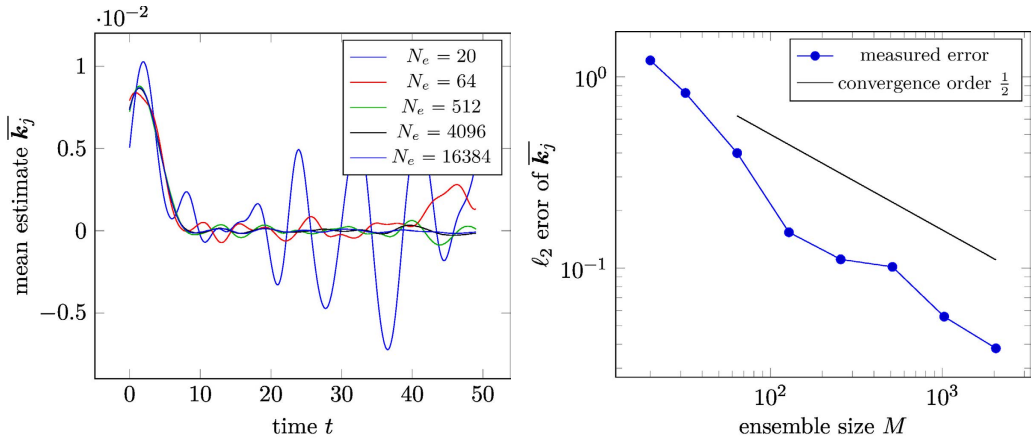
Figure 2.3: During the sequential data assimilation process, the analysis ensemble and thereby the empirical mean of the kernel function gets continuously updated, here shown for different update time steps.

previous paragraph. However, this time, we change the size of the ensemble. In Figure 2.4(a), we show the empirical mean $\overline{\boldsymbol{k}_j}$ for ensemble sizes $N_e \in \{20, 64, 512, 4096, 16384\}$. The convergence in the error of the empirical mean is shown in Figure 2.4(b). Here, we define the solution for $N_e = 16384$ as overkill solution and show convergence in the relative ℓ_2 error $\frac{\|\boldsymbol{k}_j - \boldsymbol{k}_j^{overkill}\|_{\ell_2}}{\|\boldsymbol{k}_j^{overkill}\|_{\ell_2}}$ towards this solution. The results indicate a convergence order of approximately $\frac{1}{2}$. This is the expected order of convergence, since we use a Monte Carlo-type estimator. Note that an ensemble size of about 20, which is often used for Ensemble Kalman Filters, seems not to be enough in this application. In that case, we observe a highly oscillatory result with a strong overshooting for the initial peak of the mean estimate (which will be the perfusion estimate).

2.4.4 Convergence in the time sub-steps $\Delta\tau$

In the following, we have a look at convergence with respect to the quadrature and evolution model step size $\Delta\tau$. Here, we do not use an overkill solution. To achieve this, we (discretely) fold the empirical mean $\overline{\boldsymbol{k}_j}$ against the (discretized) arterial input function $\boldsymbol{c}_{art}$, i.e. we transfer the prediction for $\boldsymbol{k}_j$ into observation space. In observation space, we compare against the analytically given artificial measurement result c. Our numerical study uses a variation of the sub-step number s, i.e. we change $\Delta\tau$ while keeping all other parameters as in Section 2.4.2.

In Figure 2.5(a), we visually compare the results obtained for an increasing number of sub-steps s (i.e. decreasing $\Delta\tau$). The convergence plot in Figure 2.5(b) further shows the error reduction in the relative ℓ_2 norm for decreasing $\Delta\tau$ if we compare the convolved mean estimate $\overline{\boldsymbol{k}_j}$ with the real observation data. The results indicate approximately first order convergence. In fact, parameter $\Delta\tau$ influences the Euler-Maruyama approximation of the continuous stochastic differential equation eq. (2.9) and the quadrature of the convolution integral. While the Euler-Maruyama method is known to have halve order convergence, the rectangular rule is convergent of second order for sufficiently smooth integrands. The observed convergence behavior strongly depends on the dominance of one of the errors (time-integration, quadrature). The



(a) ensemble estimates for $\overline{\boldsymbol{k}_j}$ (b) convergence wrt. overkill solution

Figure 2.4: With growing ensemble size, the empirical estimate for the mean of the response / kernel function gets more accurate (left) and converges with roughly order $\frac{1}{2}$ (right).

observed first order seems to indicate that the quadrature error for the convolution integral is dominant. Nevertheless, full second order convergence is not achieved. This observation is clearly a pre-asymptotic and strongly problem-dependent result.

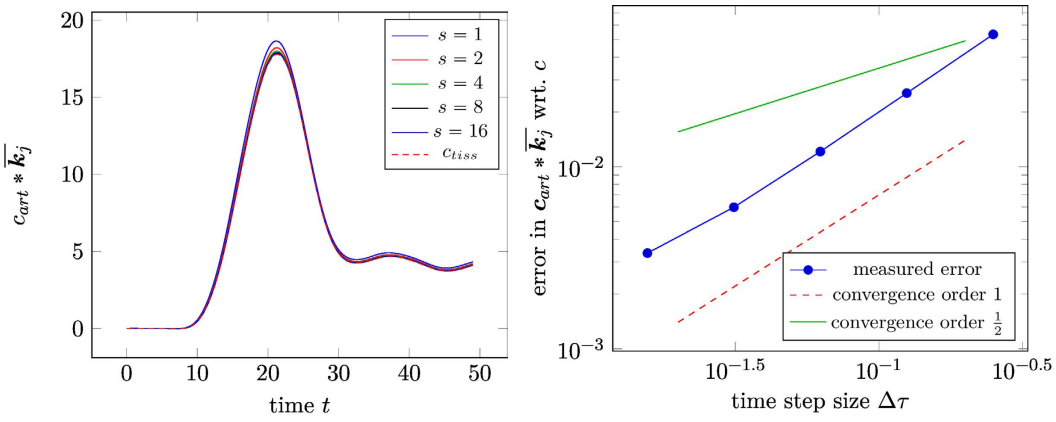
2.4.5 Influence of the correlation length in the system state noise

Our next study shall give an insight into the influence of the system state model, more specifically the influence of the correlation length ℓ of the random variable $\mathbf{n}$ on the inferred solution. To study the influence of the correlation length, we keep the parameters as in Section 2.4.2 and apply different correlation lengths $\ell \in \{0.125, 0.5, 2\}$. The results of this numerical study are given in Figure 2.6. Here, the inferred kernel function $\overline{\boldsymbol{k}_j}$ is shown for different correlation lengths. For growing correlation length the result gets less noisy. Hence, a larger correlation length has a regularizing effect on the solution. Since, in general, we seek for smooth solutions, we always choose $\ell = 2$.

2.4.6 Influence of the number of observations

Our final test with noise-free model data on a single voxel highlights the influence of a change of the observation time step size Δt_{obs}, i.e. a change in the number of observations that are made during the imaging process. To test this, we take the same parameters as in Section 2.4.2, but change the observation time step size as $\Delta t_{obs} \in \{0.125, 0.25, 0.5, 1.0\}$ while keeping $\Delta\tau$ constant. The quantity that we study is the computed probability density function for $k|_{t=0}$, hence a scaled version of $\overline{\boldsymbol{p}}$. We use the kernel density estimator `ksdensity` in Matlab to reconstruct a continuous PDF for the ensemble data.

The results of this study can be seen in Figure 2.7. Here, we make two observations. First, the mean of the PDF still changes for growing number of measurements, converging towards a true solution. Second, and more important, we observe a variance reduction if we increase



(a) $\overline{k_j}$ in observation space (b) convergence

Figure 2.5: A smaller time step size for the quadrature / system state model leads to convergence of $\overline{k_j}$ in observation space towards the measurement concentration c.

the number of measurements. This type of information would not be available in classical inverse approaches for compute perfusion estimation. That is, we can now obtain confidence information for our solution.

2.4.7 Inference from noisy data

Until now, we considered noise-free input data. Instead, we now discuss the same one-voxel input as before, but add artificial noise as

$$c_{j,i}^{noisy} = c_{j,i} + w_{j,i}\,, \quad i \in \{1, \dots, N_{obs}\}\,,$$

where the $w_{j,i}$ are realizations of i.i.d. random variables $\mathsf{w}_{j,i} : \Omega \to \mathbb{R}$, $\mathsf{w}_{j,i} \sim \mathcal{N}(0, \sigma_{\mathsf{w}})$ with $\sigma_{\mathsf{w}} \in \mathbb{R}$ the variance of the noise.

We use a series of test cases with $\sigma_{\mathsf{w}} = 10^2 \alpha_{rel}$ and $\alpha_{rel} \in \{2^{-10}, 2^{-8}, 2^{-6}, 2^{-4}, 2^{-2}\}$. Hence, α_{rel} corresponds to the relative variance with respect to the magnitude of the measurements. We keep a major part of the parameters from Section 2.4.2. However, we change the fixed observation error variance σ_{e}, to a problem-adapted one, namely, i.e. $\sigma_{\mathsf{e}} = \sigma_{\mathsf{w}}$. Note that in practice, one would empirically *estimate* the noise in the measurement data and would set σ_{e} accordingly. Another change concerns the number of samples N_e in the EnKF. As our experiments showed, the size of the ensemble has to be increased for higher variances. This is well covered by classical Monte-Carlo theory. Therefore, we set $N_e = 10000$ for $\alpha_{rel} \in \{2^{-10}, 2^{-8}, 2^{-6}\}$ while we use $N_e = 60000$ and $N_e = 100000$ for $\alpha_{rel} = 2^{-4}$ and $\alpha_{rel} = 2^{-2}$, respectively.

In Figure 2.8, we give two examples of noisy inputs for $\alpha_{rel}^2 = 0.015625$ and $\alpha_{rel}^2 = 0.0625$. In the latter case, the original input signal is already severely degenerated. The predicted mean solutions in observation space $\boldsymbol{c}_{art} * \overline{\boldsymbol{k}_j}$ are also given in Figure 2.8. In fact, the reconstructed solution is almost not influenced for $\alpha_{rel}^2 = 0.015625$ and gets a little distorted for stronger noise. In Figure 2.9, we compare estimates of the PDF for $k|_{t=0}$ for growing noise in the data.

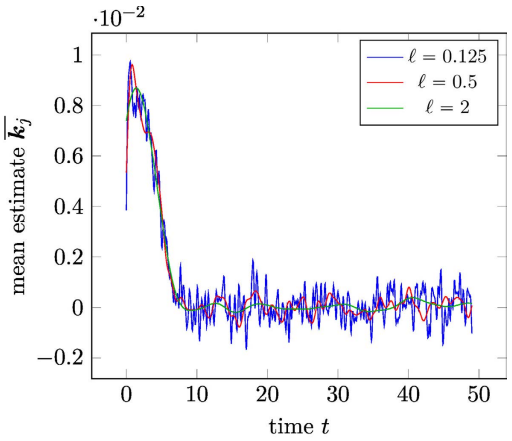


Figure 2.6: Longer correlation lengths ℓ impose a higher smoothness on the ensemble estimate.

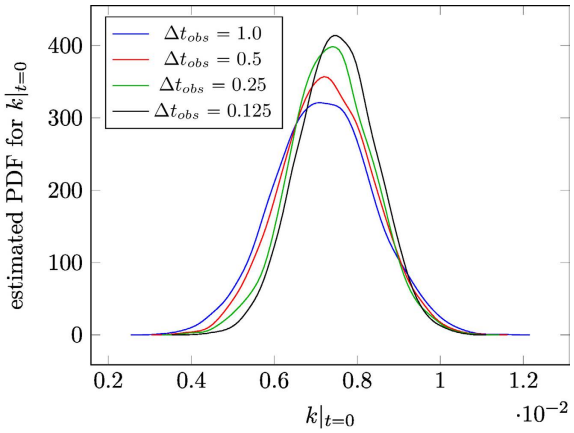
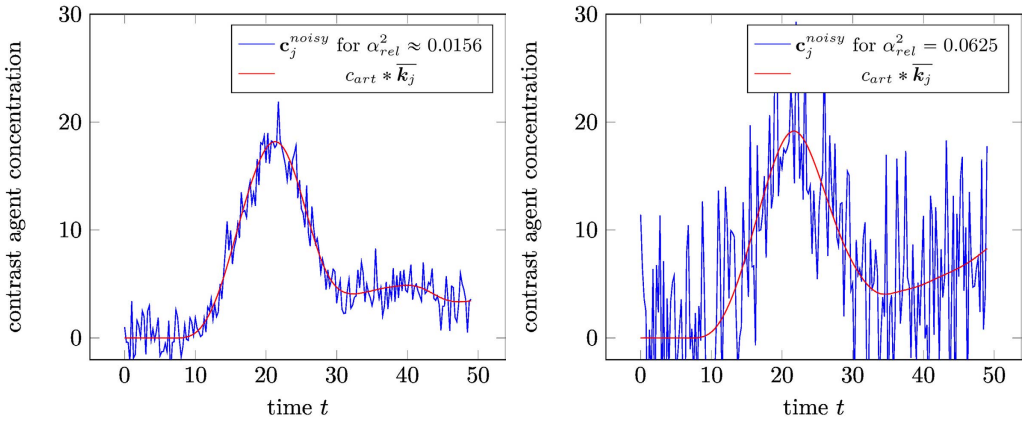


Figure 2.7: The more observation samples are taken, the more reliable the estimate of the solution. Hence, the estimated PDF for $k|_{t=0}$ shows a smaller variance for smaller observation time steps $\Delta_{t_{obs}}$.

Since we appropriately account for the noise in the input, the mean is almost identical up to $\alpha_{rel}^2 = 0.015625$. For higher relative noise variances, the probability density functions still cover the general tendency of the results. Note that the variance in the solutions grows for larger noise in the input. This effect is not primarily caused by the noisy input, but by the imposed observation error $\mathsf{e}_{j,i}$, which acts here as a regularization for the noisy input. Nonetheless, as long as the observation error variance is set in the range of the input noise variance, the variance in the solution correctly represents the variance coming from the noise in the input.

2.4.8 Application problem

We finally apply the beforehand studied method to a full application problem given by the Digital Brain Perfusion Phantom introduced in Section 2.4. We use the slice $(\cdot, \cdot, 150)$ with the choice of regions with reduced and severely reduced perfusion as in Figure 2.1. We discuss



(a) (b)

Figure 2.8: Even for stronger noise on the input data the inference of $\overline{\boldsymbol{k}_j}$ is acceptable, as long as the measurement variance σ_{e} is chosen appropriately. Here, we compare the noisy input $\boldsymbol{c}_{\boldsymbol{j}}^{noisy}$ $\overline{\boldsymbol{k}_j}$ in observation space for $\alpha_{rel}^2 = 0.015625$ (left) and $\alpha_{rel}^2 = 0.0625$ (right).

a result for a large observation time-step size $\Delta t_{obs} = 1.0$, a highly resolved quadrature with $\Delta\tau = 0.0625$ and a noise with variance $\sigma_{\mathsf{w}} = 10^2\, 0.015625$. The observation error variance is adapted as $\sigma_{\mathsf{e}} = \sigma_{\mathsf{w}}$. All other parameters are kept as in Section 2.4.2.

Storage and performance considerations

Storing and computing the ensembles for the discussed test cases is a rather challenging task. Just considering the analysis ensemble for a single slice, we need to store for each of the 256×256 voxels 5000 realizations of discrete kernel functions $\boldsymbol{k}_j$ given via $N_q = 785$ double precision values leading to a total storage requirement of

$$256 \times 256 \times 5000 \times 785 \times 8 \text{ Bytes} \approx 1834 \text{ GBytes}\,.$$

All our calculations are done in Matlab. We always compute 8 rows of the final 256×256 slice at the same time and reuse the random input for each voxel in order to reduce the runtime. Note that especially sampling from $\mathbf{n}^{(n)}$ is very computationally demanding. In order to do the calculations, we need constant access to way more than 64 GBytes of RAM. Due to storage und memory requirements, we use nodes of the cluster *Rhea* at Oak Ridge National Lab to compute the full problem. Each node has 128 GBytes of RAM and a dual Intelő Xeonő E5-2650 CPU with 16 cores. To compute 8 lines, i.e. results for $8 \times 256 = 2048$ voxels, we need about 3 hours and 15 minutes, noting that Matlab uses approximately 14 cores of the full machine. The total computing time (with respect to one node of Rhea) is thereby roughly 104 hours or about 4.3 days on a single machine.

Even though this amount of computing time seems to be rather prohibitive for the specific application case, it is clear that the discussed algorithm is extremely easy to parallelize. Espe-

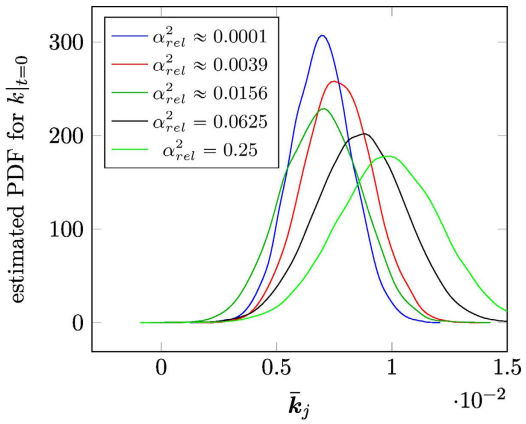
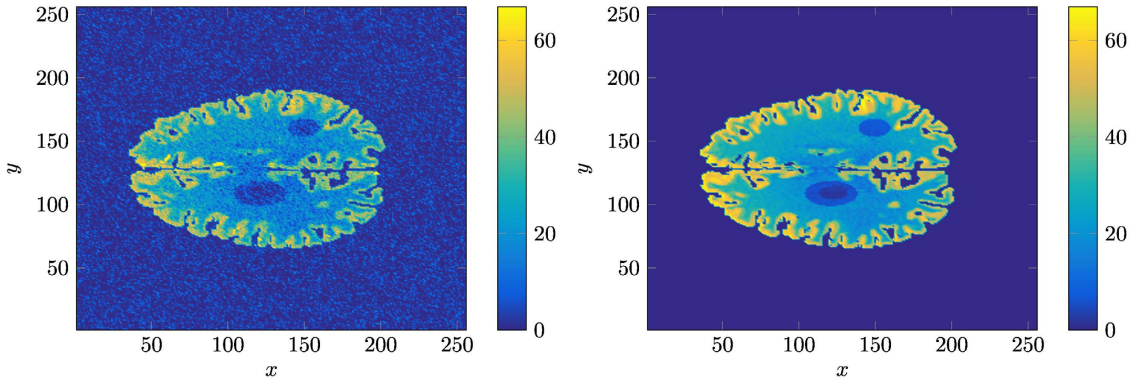


Figure 2.9: The proposed method is pretty robust with respect to noise. This can be seen, if we study the estimated probability density functions for $k|_{t=0}$. With growing noise variance, the empirical PDF estimate still recovers the mean appropriately. Extreme noise variances degenerate the result, as expected.

cially, it seems to be very well suited to a parallelization on graphics processing units (GPUs) or other many-core hardware, as long as the results of the calculation are constantly streamed out to CPU memory. An appropriate parallel implementation is future work.



(a) Approximated perfusion $\overline{\boldsymbol{p}}$

(b) Noise-free reference perfusion given by the DPP

Figure 2.10: Our approximation method recovers the reference perfusion result (right) as mean of the ensemble in the Ensemble Kalman Filter. Both results match well, even though we introduced a considerable amount of artificial noise.

Quantities of interest

In our application examples, we consider the approximation of probabilistic quantities of interest in connection with the perfusion $p_j := p(k_j) = \frac{1}{\rho_j} k(0)$. Besides of the mean $\overline{p_j}$ we are especially interested in probabilities for the corresponding random variable p_j to be in a given

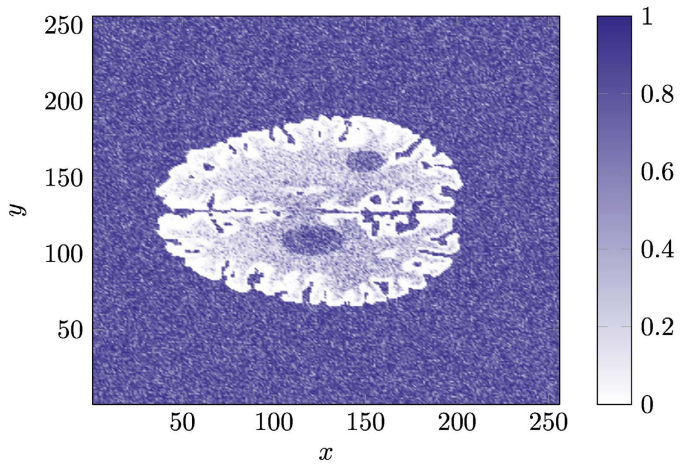


Figure 2.11: The advantage of the proposed method is that we are now also able to compute probabilistic information for the solution, here shown by plotting the probability $\mathbb{P}(\mathsf{p}_j < 10)$. Hence, the depicted results give the space-dependent probability for low (< 10) perfusion.

range. To be more specific, we compute the probabilities

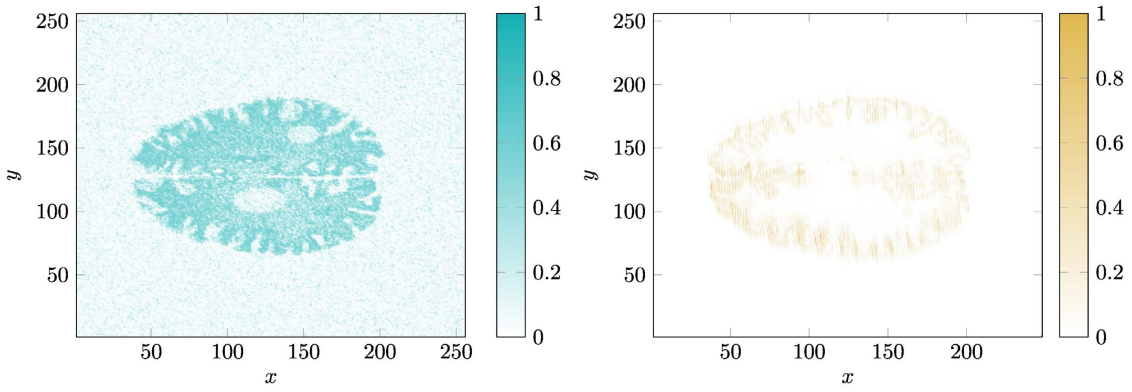
$$\mathbb{P}(\mathsf{p}_j < 10)\,, \quad \mathbb{P}(20 \leqslant \mathsf{p}_j < 40)\,, \quad \mathbb{P}(\mathsf{p}_j \geqslant 50)\,,$$

noting that the underlying perfusion p_j lies in the interval $[0, 70]$ in the case of the Digital Brain Perfusion Phantom data that we consider. These quantities give probabilities for low, medium and high perfusion in some region of the brain. Given the final analysis ensemble for k_j, it is easy to compute the above quantities by using the kernel density estimator `ksdensity`. The latter one can compute a cumulative distribution function (CDF) for each voxel, which is finally evaluated appropriately.

Discussion of results

An important advantage of the use of a Digital Perfusion Phantom is the existence of a reference solution to compare with. The DPP software that we use stores the reference solution together with the other generated data. In Figure 2.10(b), we show the reference solution for our full application test case. The approximated result of our application example study, i.e. $\overline{\boldsymbol{p}}$, is shown in Figure 2.10(a). As expected from our single-voxel study, the inferred perfusion matches the exact perfusion result well. Note that this is the case even though we add a considerable amount of noise on the measurements.

As discussed before, we can use the ensemble-based estimate of the posterior probability density function to extract a wide range of probabilistic information on the inferred solution. This is the main result of this work. To exemplify this, we compute space-dependent probabilities for low (Figure 2.11), medium (Figure 2.12(a)) and high (Figure 2.12(b)) perfusion ranges, cf. Section 2.4.8. In case of Figure 2.11, we e.g. can now easily identify ranges of low perfusion and even can give a probability for this result.



(a) $\mathbb{P}(20 \leqslant \mathsf{p}_j < 40)$ (b) $\mathbb{P}(\mathsf{p}_j \geqslant 50)$

Figure 2.12: Based on the results of the EnKF, it is easily possible to identify regions of high probability to have medium (left) and high (right) perfusion.

In general, we claim that this probability information or derived probabilistic quantities (variance, percentiles, etc.) can give domain-experts in radiology a much clearer information on the reliability of the inferred estimates.

2.5 Summary

In this work, we have discussed the use of Ensemble Kalman Filters for sequential data assimilation in order to infer probabilistic information on (blood) perfusion in tissue for given measurements from dynamic contrast–enhanced imaging. The deterministic inference of perfusion is well-known in the field of radiological imaging. However, to the best of the author's knowledge, the new contribution is the approximation of PDFs for the perfusion given (noisy) measurements. EnKF are well-known in inference for dynamical systems and partial differential equations with stochastic coefficients. Hence, modeling the dynamic contrast–enhanced imaging process as sequential data assimilation in a Bayesian context was the main contribution of the work. Given the ensemble-based approximation of the PDF, we could compute probabilistic quantities such as probabilities for perfusion parameter ranges.

The new approach was first investigated for a single-voxel example with respect to convergence and parameter influence. Afterwards, it was applied to artificial application data generated by a Digital Perfusion Phantom, i.e. a model for deriving DCE image data for given perfusion data. Overall, the effectiveness of the method could be demonstrated, showing empirical convergence results and appropriate approximations of probabilistic information. The use of realistic patient data, refined problem-adapted covariance kernels, advanced filtering techniques and an efficient parallel implementation are future work.

Acknowledgements

This work is funded by the Swiss National Science Foundation (SNF) under project number 407540_167186. Furthermore, this research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

The author also likes to thank Wolfram Stiller and Christian Weis of the department of *Diagnostic and Interventional Radiology* of the *University Medical Center Heidelberg* and Holger Fröning of the *Institute of Computer Engineering at University of Heidelberg* for fruitful initial discussions on the application background.

References

[And01] J. L. Anderson. An ensemble adjustment Kalman filter for data assimilation. *Monthly weather review*, 129(12):2884–2903, 2001.

[And03] J. L. Anderson. A local least squares framework for ensemble filtering. *Monthly Weather Review*, 131(4):634–642, 2003.

[BGKW10] G. Brix, J. Griebel, F. Kiessling, and F. Wenz. Tracer kinetic modelling of tumour angiogenesis based on dynamic contrast-enhanced CT and MRI measurements. *European journal of nuclear medicine and molecular imaging*, 37(1):30–51, 2010.

[ESS15] O. G. Ernst, B. Sprungk, and H.-J. Starkloff. Analysis of the ensemble and polynomial chaos Kalman filters in Bayesian inverse problems. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):823–851, 2015.

[Eve09] G. Evensen. *Data assimilation: the ensemble Kalman filter*. Springer Science & Business Media, 2009.

[FKG+11] A. Fieselmann, M. Kowarschik, A. Ganguly, J. Hornegger, and R. Fahrig. Deconvolution-based CT and MR brain perfusion measurement: theoretical model revisited and practical implementation details. *Journal of Biomedical Imaging*, 2011:14, 2011.

[ILS13] M. A. Iglesias, K. J. Law, and A. M. Stuart. Ensemble Kalman methods for inverse problems. *Inverse Problems*, 29(4):045001, 2013.

[K+60] R. E. Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[Man] M. Manhart. *Digital Brain Perfusion Phantom Documentation*. provided on the data web page of the pattern recognition lab at FAU Erlangen-Nürnberg, Germany (last check: Feb. 23, 2017).

[MT18] A. J. Majda and X. T. Tong. Performance of ensemble Kalman filters in large dimensions. *Communications on Pure and Applied Mathematics*, 71(5):892–937, 2018.

[NSL16] G. Nævdal, O. Sævareid, and R.-J. Lorentzen. Data assimilation using MRI data. In *Proceedings of the ECCOMAS Congress 2016*, 2016.

[ØWC+96] L. Østergaard, R. M. Weisskoff, D. A. Chesler, C. Gyldensted, and B. R. Rosen. High resolution measurement of cerebral blood flow using intravascular tracer bolus passages. Part I: Mathematical approach and statistical analysis. *Magnetic resonance in medicine*, 36(5):715–725, 1996.

[Pia12] O. S. Pianykh. Digital perfusion phantoms for visual perfusion validation. *American Journal of Roentgenology*, 199(3):627–634, 2012.

[PRM+06] G. J. Parker, C. Roberts, A. Macdonald, G. A. Buonaccorsi, S. Cheung, D. L. Buckley, A. Jackson, Y. Watson, K. Davies, and G. C. Jayson. Experimentally-derived functional form for a population-averaged high-temporal-resolution arterial input function for dynamic contrast-enhanced mri. *Magnetic resonance in medicine*, 56(5):993–1000, 2006.

[RC13] S. Reich and C. J. Cotter. Ensemble filter techniques for intermittent data assimilation. *Large Scale Inverse Problems. Computational Methods and Applications in the Earth Sciences*, 13:91–134, 2013.

[RC15] S. Reich and C. Cotter. *Probabilistic forecasting and Bayesian data assimilation.* Cambridge University Press, 2015.

[Rei13] S. Reich. A nonparametric ensemble transform method for Bayesian inference. *SIAM Journal on Scientific Computing*, 35(4):A2013–A2024, 2013.

[RPV+11] A. J. Riordan, M. Prokop, M. A. Viergever, J. W. Dankbaar, E. J. Smit, and H. W. de Jong. Validation of CT brain perfusion methods using a realistic dynamic head phantom. *Medical physics*, 38(6):3212–3221, 2011.

[Sou14] S. Sourbron. A tracer-kinetic field theory for medical imaging. *IEEE transactions on medical imaging*, 33(4):935–946, 2014.

[SS17] C. Schillings and A. M. Stuart. Analysis of the ensemble Kalman filter for inverse problems. *SIAM Journal on Numerical Analysis*, 55(3):1264–1290, 2017.

[SS18] C. Schillings and A. M. Stuart. Convergence analysis of ensemble Kalman inversion: the linear, noisy case. *Applicable Analysis. An International Journal*, 97(1):107–123, 2018.

[Stu10] A. M. Stuart. Inverse problems: a Bayesian perspective. *Acta Numerica*, 19:451–559, 2010.

[TAB+03] M. K. Tippett, J. L. Anderson, C. H. Bishop, T. M. Hamill, and J. S. Whitaker. Ensemble square root filters. *Monthly Weather Review*, 131(7):1485–1490, 2003.

[Tof97] P. S. Tofts. Modeling tracer kinetics in dynamic Gd-DTPA MR imaging. *Journal of Magnetic Resonance Imaging*, 7(1):91–101, 1997.

[Ton18] X. T. Tong. Performance Analysis of Local Ensemble Kalman Filter. *Journal of Nonlinear Science*, 28(4):1397–1442, 2018.

[ZZH09] F. Zhang, M. Zhang, and J. A. Hansen. Coupling ensemble Kalman filter with four-dimensional variational data assimilation. *Advances in Atmospheric Sciences*, 26(1):1–8, 2009.

3 Subspace correction methods in algebraic multi-level frames

3.1 Introduction

The solution of large sparse linear systems is the dominant computational part of many solvers for partial differential equations (PDEs). Multigrid solvers are optimal problem-size independent iterative solvers for such systems. Nevertheless, multigrid solvers tend to have limited parallel scalability on extreme-scale high performance computing (HPC) clusters [BFG+12]. Moreover, next generation Exascale clusters are expected to have a growing number of (hardware) failures [CGG+14]. These issues require us to develop new optimal-complexity linear solvers that are scalable and resilient.

Randomized subspace correction iterative solvers for multi-level frames, cf. [GO12], give rise to mesh-width independent solvers for e.g. elliptic problems. The randomized, greedy-type solution algorithm from [GO12] is error-resilient by construction and might expose a high grade of parallelism by some suitable extension. However, it has been only discussed for model-type problems, since knowledge of the geometric structure of the discretized problem geometry is needed for the multi-level construction.

To easily solve problems on complex geometries, we introduce a purely algebraic multi-level construction approach. The multi-level construction is based on classical (Ruge-Stüben) algebraic multigrid (AMG) [Stü01, RS86]. Projections between grid levels are replaced by algebraic prolongations and restrictions, while the different levels are generated by standard AMG coarsening. The solution of the resulting algebraic multi-level frame linear system by a Gauss-Seidel iterative method is equivalent to some algebraic multigrid method with Gauss-Seidel smoother. Moreover, we construct a new randomized greedy-type algebraic multi-level method by applying randomized greedy subspace correction techniques from [GO12] to the algebraic multi-level frame system.

Numerical results will be given, showing that the new method matches the geometry-dependent solver results from [GO12] for equivalent complex-geometry elliptic problems discretized by linear finite elements. The purely algebraic construction makes the proposed method an optimal candidate for general-purpose linear solver libraries. Resilience and potential massive parallelism might be influencal for future linear solver developments in Exascale computing.

In Section 3.2, we discuss related work. Section 3.3 reviews standard (geometric) multi-level frames with iterative solvers and subspace correction schemes. Section 3.4 introduces the new algebraic multi-level frames covering the necessary AMG background. Numerical results are given in Section 3.5. Section 3.6 concludes this study with a short outlook.

3.2 Related work

Theory and numerical treatment of the so-called *multi-level generating system* for (elliptic) PDEs has been first discussed in [Gri93, Gri94] and a series on follow-up studies [GO12, GO93, GO95b, GHO15]. In the generating system approach, a basis for a fixed Galerkin finite element discretization level is replaced by a generating system of multi-level basis functions. Equivalence of the application of standard iterative solvers to the resulting multi-level generating system and optimal-complexity multi-level solvers on single-level discretizations has been intensively discussed in [Gri93, Gri94]. More recently, multi-level generating systems have been reconsidered as *multi-level frames* [DFR$^+$04, HSS08], accounting for the close connection to wavelet-based techniques [Dah97]. Many of these approaches can be also re-interpreted as additive or multiplicative Schwarz methods [GO12, GO95a].

The application of more sophisticated iterative solvers to multi-level frame or generating system discretizations naturally leads to new classes of multi-grid/-level type algorithms. In [GO12, The10], the Gauss-Southwell method [Sou40] has been applied in this context, giving rise to a new greedy-type multi-level method. The new method shows problem-size independent convergence with a high potential for robustness and problem-adaptivity. Moreover, a preceding randomized subspace selection process leads to a randomized iterative subspace correction technique with optimal convergence. This randomized iterative subspace correction technique has a close relationship to randomized (block) Kaczmarz iterative methods [ZF13, OZ] and randomized coordinate descent methods [Nes12, BT13, Mai13], which are known from large-scale optimization. Another related field are asynchronous iteration techniques [FS00]. These techniques provide a theoretical and practical framework for iterative methods that allow for the re-use of iteration results from several previous steps. Indeed, they allow to define asynchronous update steps in a single iterative process. Therefore, recent applications of asynchronous iterations are scalable and hardware-aware iterative solvers and multigrid smoothers [ATG$^+$12, ATDH13].

Randomized and asynchronous iterative methods promise to overcome *resilience* issues, which are expected to occur for HPC clusters of growing size [CGG$^+$14]. While iterative methods are generally well-known to be robust for a certain class of hardware- or software-induced errors, resilient multigrid methods are still subject of current research [CdSBS12, HGRW15]. *Scalability* of multigrid methods and smoothers might also be improved by randomized and asynchronous iterative methods. Here, recent studies, such as [BFG$^+$12], suggest a scalability bottleneck for standard smoothers on large-scale HPC systems. In addition, *problem-adapted* multigrid techniques such as semi-coarsening [DIR92] and line smoothers [OGWW98] have been proposed before. They require some geometric knowledge of the problem. In contrast, purely algebraic greedy-type approaches on multi-level frames promise to deliver optimal problem-dependent iterative traversal strategies that go across all available discretization levels. These greedy approaches might therefore lead to fast convergence for a much wider class of problems.

The algebraic or matrix-dependent construction of multi-level or multigrid methods is attractive for optimal-complexity black-box linear algebra libraries. A review of these methods is given in [Stü01]. More recently, two techniques, namely classical Ruge-Stüben AMG [RS86] with further developments e.g. in [GMOS06, Met13] and (smoothed) aggregation-based AMG [VMB96] are in main use. While classical AMG constructs hierarchies of variables by *reusing* a subset of the variables from a fine level on the next coarser level, aggregation-based techniques

construct new variables on a coarser level by *replacing* a set of fine grid variables. Aggregation-based techniques tend to be easier to implement and parallelize. On the other hand, classical AMG is more robust.

To the best of our knowledge, the only known connection between AMG and frame-based constructions has been discussed in the partially unpublished work [Kie01, GK01, GKK03], where so-called AMGlets, thus algebraically constructed wavelets, are considered. AMGlets primarily aim at generalizing basis function constructions for some classes of differential operators, while the present work focuses on geometry-independent optimal solvers. Greedy-based techniques have been discussed for AMG with respect to the construction of the multigrid hierarchy [MS07], but not for smoothers, so far. Therefore, we argue that the introduction of an AMG-based multi-level frame technique is new. Moreover, by transferring randomized subspace correction methods to AMG, we introduce a new class of resilient, scalable and problem-adapted algebraic multi-level methods.

3.3 Multi-level frame systems and their iterative solution

This section introduces the *original* (geometric) multi-level frame construction. A short review of multi-level frames for the solution of elliptic PDEs is given. Some knowledge of the structure of the resulting linear system is collected. Finally, properties of iterative linear solvers for multi-level frame systems are summarized.

3.3.1 Multi-level frames for the solution of elliptic PDEs

We closely follow [GO12, HSS08] for the introduction of multi-level frame discretizations of elliptic PDEs.

For a separable Hilbert space $\mathcal{H}$, its dual space $\mathcal{H}'$ and the duality product $< \cdot, \cdot >$ on $(\mathcal{H}, \mathcal{H}')$, we can introduce a *frame* for $\mathcal{H}$ as the countable collection $\Phi = \{\phi_i : i \in \Delta\} \subset \mathcal{H}$ with

$$C\|f\|_{\mathcal{H}'}^2 \leqslant \sum_{i\in\Delta} |\langle f, \phi_i\rangle|^2 \leqslant D\|f\|_{\mathcal{H}'}^2 , \qquad \text{for all } f \in \mathcal{H}' .$$

Let $\Omega \subset \mathbb{R}^{\{2,3\}}$ be a piecewise linearly bounded domain. As a model problem, we aim at solving the elliptic partial differential equation

$$\begin{aligned} -\Delta u &= g \quad \text{in } \Omega , \\ u &= 0 \quad \text{on } \partial\Omega , \end{aligned}$$

with $u, g : \Omega \to \mathbb{R}$, by a Galerkin approach. We therefore choose $\mathcal{H} = H_0^1(\Omega)$ and introduce the usual bilinear form

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, dx, \quad a : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$$

and the corresponding linear functional

$$F(v) = \int_\Omega gv \, dx, \quad F \in \mathcal{H}' .$$

For a standard linear finite element discretization, we introduce an appropriate triangulation or partition $\mathcal{T}_l$, $l \geqslant 0$ of Ω with element sizes $h \approx 2^{-l}$. Then,

$$V_l := \operatorname{span}\{\varphi_{l,k} | k \in \Delta_l\}$$

is the finite-dimensional subspace of linear Lagrange finite element basis functions over $\mathcal{T}_l$. Finally, we have to solve the linear system

$$A_l \boldsymbol{u}_l = \boldsymbol{f}_l$$

with

$$A_l = (a_{l;k,k'})_{k,k'}, \quad a_{l;k,k'} := a(\varphi_{l,k}, \varphi_{l,k'}) \quad \text{and} \quad f_{l;k} = F(\varphi_{l,k})$$

in order to get an approximation for the given PDE in V_l.

For a *multi-level frame* approximation, we instead introduce the *sequence* of nested subspaces

$$V_0 \subset V_1 \subset \ldots \subset V_l \subset \ldots \subset L^2(\Omega)$$

for nested triangulations. For geometric multi-level frames, the coarsest subspace V_0 is usually expected to resolve the boundary exactly, which might be a strong limitation.

The objective of multi-level frames is the approximation of the PDE as in a multi-resolution analysis. That is, we introduce the (countable) collection

$$\Phi = \{\varphi_{l,k} | k \in \Delta_l, l \in \mathbb{N}_0\},$$

for which we further assume each function $\varphi_{l,k}$ to be normalized with respect to the $H^1(\Omega)$-norm. Following [HSS08, Theorem 5], this collection defines a frame in $H^1(\Omega)$. By replacing the standard finite element basis with the (finite) collection or *generating system*

$$\Phi_L = \{\varphi_{l,k} | k \in \Delta_l, l \in \{0, \ldots, L\}\}$$

of maximum level L, cf. [GO12], we are able to introduce a multi-level discretization on level L, with a new linear system

$$\bar{A}_L \bar{\boldsymbol{u}}_L = \bar{\boldsymbol{f}}_L \,. \tag{3.1}$$

Here, the system matrix is composed of matrix blocks as

$$\bar{A}_L = \left(A_{l,l'}\right)_{1 \leqslant l,l' \leqslant L}, \quad A_{l,l'} = \left(a_{l,l';k,k'}\right)_{k \in \Delta_l, k' \in \Delta_{l'}}, \quad a_{l,l';k,k'} = a(\varphi_{l,k}, \varphi_{l',k'})$$

and the right-hand side is given as

$$\bar{\boldsymbol{f}}_L = (\boldsymbol{f}_l)_{1 \leqslant l \leqslant L}, \quad \boldsymbol{f}_l = (f_{l;k})_{k \in \Delta_l}, \quad f_{l;k} = F(\varphi_{l,k}) \,.$$

By construction, the matrix of the multi-level frame system (3.1) is singular. However, cf. [HSS08], the right-hand side lies in the image of the multi-level system matrix. For such problems, Krylov subspace solvers converge to a (non-unique) solution vector $\bar{\boldsymbol{u}}^L$. The final unique solution on level L is constructed by projection on the solution space V_L.

Algorithm 2 Gauss-Seidel method

Require: $A \in \mathbb{R}^{N\times N}$, $\boldsymbol{b} \in \mathbb{R}^N$, $\boldsymbol{x}_{init} \in \mathbb{R}^N$, N_{iter}
1: **function** GaussSeidel
2: $\quad \boldsymbol{x}^{(0)} := \boldsymbol{x}_{init}$
3: $\quad$ **for** $n \in \{0, \dots, N_{iter} - 1\}$ **do**
4: $\quad\quad$ **for** $i \in \{1, \dots, N\}$ **do**
5: $\quad\quad\quad \boldsymbol{x}_i^{(n+1)} := \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(n+1)} - \sum_{j=i+1}^{N} a_{ij} x_j^{(n)} \right)$
6: $\quad$ **return** $\boldsymbol{x}^{(N_{iter})}$

3.3.2 Structure of the multi-level system

In [Gri93, The10], we learned that the multi-level system (3.1) has some specific structure. We can easily rewrite it in terms of the standard finite element stiffness matrix on level L and appropriate prolongation and restriction operators P_l and R_l that transfer functions between the different approximation spaces V_l as

$$P_l : V_l \to V_{l+1} \quad \text{and} \quad R_l : V_l \to V_{l-1} \,.$$

In the following, these linear operators are understood as matrices with respect to the appropriate basis of the individual level. For the standard multi-level frame construction, P_l and R_l are chosen as in an appropriate standard geometric multigrid method. We also use the more general notation $P_l^{l+1} := P_l$ and $P_l^{l-1} := R_l$.

The reformulation of the multi-level problem (3.1) further involves to introduce the transfer matrices

$$S_L := \begin{bmatrix} P_L^0 \\ P_L^1 \\ \vdots \\ P_L^{L-1} \\ P_L^L \end{bmatrix}, \quad S^L := \begin{bmatrix} P_0^L P_1^L \dots P_{L-1}^L P_L^L \end{bmatrix}$$

with concatenated restriction operators $P_L^l = P_{l+1}^l \cdot \ldots \cdot P_L^{L-1}$ and $P_L^L = I_L$ the identity matrix on level L. We can then rewrite (3.1) as

$$\bar{A}_L \bar{\boldsymbol{u}}_L = S_L A_L S^L \bar{\boldsymbol{u}}^L = S_L \boldsymbol{f}_L = \bar{\boldsymbol{f}}_L \,. \tag{3.2}$$

A solution on level L can be derived from the multi-level solution $\bar{\boldsymbol{u}}_L$ by

$$\boldsymbol{u}_L = S^L \bar{\boldsymbol{u}}_L \,.$$

3.3.3 Iterative solvers on multi-level frame systems

Following [Gri94], the iterative solution of the multi-level system (3.1) for a maximum level of L corresponds to the solution of the standard single-level finite element problem on level L by a sophisticated, typically optimally preconditioned, iterative solver.

Solving the system (3.1) by a Jacobi-preconditioned conjugate gradient iterative solver cor-

Algorithm 3 Gauss-Southwell method

Require: $A \in \mathbb{R}^{N\times N}$, $\boldsymbol{b} \in \mathbb{R}^N$, $\boldsymbol{x}_{init} \in \mathbb{R}^N$, N_{iter}

1: **function** GaussSouthwell
2: $\quad \boldsymbol{x}^{(0)} := \boldsymbol{x}_{init}$
3: $\quad$ **for** $n' \in \{0, \dots, NN_{iter} - 1\}$ **do**
4: $\quad\quad \boldsymbol{r}^{(n')} := \boldsymbol{b} - A\boldsymbol{x}^{(n')}$
5: $\quad\quad i^\star := \underset{i\in\{1,\dots,N\}}{\operatorname{argmax}} \left| r_i^{(n')} \right|$
6: $\quad\quad x_{i^\star}^{(n'+1)} := \frac{1}{a_{i^\star i^\star}} \left(b_{i^\star} - \sum_{j=1, j\neq i^\star}^{N} a_{i^\star j} x_j^{(n')} \right)$
7: $\quad$ **return** $\boldsymbol{x}^{(NN_{iter})}$

responds to a BPX-preconditioned CG solver for the standard problem. Furthermore, the standard Gauss-Seidel iteration for the linear system $A\boldsymbol{x} = \boldsymbol{b}$, $A \in \mathbb{R}^{N\times N}$, which is given in Algorithm 2, is equivalent to some standard geometric multigrid V-cycle with Gauss-Seidel smoother [Gri94, GO12].

In [Sou40], a greedy version of the Gauss-Seidel method was introduced. It is called *Gauss-Southwell* method. For a given iterate $\boldsymbol{x}^{(n')}$, it computes the residual $\boldsymbol{r}^{(n')} = \boldsymbol{b} - A\boldsymbol{x}^{(n')}$. Here, it picks the variable index $i^\star$ with maximum absolute residual

$$i^\star := \underset{i\in\{1,\dots,N\}}{\operatorname{argmax}} \left| r_i^{(n')} \right| .$$

Then, a correction step is applied such that we have a zero residual for variable $x_{i^\star}^{(n'+1)}$ in the next iteration. Algorithm 3 summarizes the Gauss-Southwell method.

Since the Gauss-Southwell method is not well-known, we include a theoretical convergence result in our discussion. In [GO12], theory of iterative subspace correction schemes based on stable space splittings is used to show a theoretical relative error reduction result for the Gauss-Southwell method in case of the solution of variational problems. We here use a remark in [GO12] that allows to simplify and apply the given result to Algorithm 3 with the requirement on A to be a symmetric positive definite matrix. Then we get the relative error reduction in the energy norm as

$$\begin{aligned} \|\boldsymbol{x} - \boldsymbol{x}^{(n')}\|_A^2 &\leqslant \left(1 - \tfrac{\lambda_{min}(A)}{\operatorname{tr}(A)}\right)^{n'} \|\boldsymbol{x} - \boldsymbol{x}_{init}\|_A^2 \\ &\leqslant \left(1 - \tfrac{1}{N\kappa(A)}\right)^{n'} \|\boldsymbol{x} - \boldsymbol{x}_{init}\|_A^2 , \qquad \text{for all } n' \geqslant 1 , \end{aligned}$$

with $\kappa(A)$ the classical spectral condition number of the system matrix A, $\lambda_{min}(A)$ its smallest eingenvalue, $\operatorname{tr}(A)$ the trace, and $\boldsymbol{x}$ the exact solution of the linear system $A\boldsymbol{x} = \boldsymbol{b}$.

From a computational point of view, [The10] gives an up to $O(\log N)$ complexity technique for one single step of the Gauss-Southwell method. A total of N of these one-variable-update steps is usually considered as one iteration, cf. Algorithm 3. In [GO12] and [The10], it is shown empirically that an iterative solution of the multi-level frame system by the Gauss-Southwell method is at least as fast as the Gauss-Seidel method, in terms of iterations. In many cases,

Algorithm 4 k-random block-Gauss-Seidel method

Require: $A \in \mathbb{R}^{N\times N}$, $\boldsymbol{b} \in \mathbb{R}^N$, $\boldsymbol{x}_{init} \in \mathbb{R}^N$, N_{iter}, k

function kRandomBlockGaussSeidel
  $\boldsymbol{x}^{(0)} := \boldsymbol{x}_{init}$
  for $n' \in \{0, \dots, NN_{iter} - 1\}$ **do**
    $\boldsymbol{r}^{(n')} := \boldsymbol{b} - A\boldsymbol{x}^{(n')}$
    $I :=$ uniformRandomSubset $(k, \{1, \dots, N\})$
    $i^\star := \operatorname{argmax}_{i\in I} \left| r_i^{(n')} \right|$
    $\boldsymbol{x}_{i^\star}^{(n'+1)} := \frac{1}{a_{i^\star i^\star}} \left(b_{i^\star} - \sum_{j=1, j\neq i^\star}^{N} a_{i^\star j} x_j^{(n')} \right)$
  return $\boldsymbol{x}^{(NN_{iter})}$

the Gauss-Southwell algorithm clearly outperforms the Gauss-Seidel method.

Another approach discussed in [GO12] is a modification of the Gauss-Southwell method. The modified approach *randomly* picks a new variable index instead of optimizing over all variables. We call this approach *random Gauss-Seidel* method and assume an equal distribution over all variables. By restricting theoretical results from [GO12] to the case of linear systems with symmetric positive definite system matrix, we can give the expected relative error reduction for the random Gauss-Seidel method as

$$\begin{aligned} \mathbb{E}\left(\|\boldsymbol{x} - \boldsymbol{x}^{(n')}\|_A^2 \right) &\leqslant \left(1 - \frac{\lambda_{min}(A)}{\operatorname{tr}(A)} \right)^{n'} \|\boldsymbol{x} - \boldsymbol{x}_{init}\|_A^2 \\ &\leqslant \left(1 - \frac{1}{N\kappa(A)} \right)^{n'} \|\boldsymbol{x} - \boldsymbol{x}_{init}\|_A^2 , \qquad \text{for all } n' \geqslant 1 , \end{aligned}$$

Numerical results in [GO12] show similar (optimal) solution performance for the multi-level frame system, in terms of asymptotic complexity, but a larger constant.

This limitation is removed by an alternative approach discussed in [GO12], which we call *k-random block-Gauss-Seidel* method. It randomly picks k variables and performs the greedy Gauss-Southwell-type correction on that variable subset, cf. Algorithm 4. In [GO12] and [OZ] a theoretical error reduction result is given for a slightly modified version of Algorithm 4. The modified algorithm updates all randomly chosen variables in subset I at once, in an additive Schwarz correction fashion. Therefore, the optimization step is neglected. For the case of a symmetric positive definite system matrix A, the theoretical error reduction is in this case

$$\|\boldsymbol{x} - \boldsymbol{x}^{(n')}\|_A^2 \leqslant \left(1 - \frac{k}{N\kappa(A)} \right)^{n'} \|\boldsymbol{x} - \boldsymbol{x}_{init}\|_A^2 , \qquad \text{for all } n' \geqslant 1 .$$

Coming back to the greedy optimization setting of Algorithm 4, [GO12] shows numerical results for for the k-random block-Gauss-Seidel method applied to a multi-level frame discretization of an elliptic problem. These results suggest that the k-random block-Gauss-Seidel method outperforms the standard Gauss-Seidel method for $k \geqslant 3$. Note that for $k = 1$, Algorithm 4 is identical to the random Gauss-Seidel method.

3.4 Algebraic multi-level frames

In our new purely algebraic approach, we aim at solving general linear systems of type

$$A\boldsymbol{x} = \boldsymbol{b}$$

with $A \in \mathbb{R}^{N\times N}$ and $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^N$ by a multi-level method. We further require A to be an M-matrix, to be able to apply classical AMG in its original form. Our aim is then to introduce an algebraically constructed variable hierarchy $\mathcal{D}_0 \subset \mathcal{D}_1 \subset \ldots \subset \mathcal{D}_L$ with $\mathcal{D}_L := \{1, \ldots, N\}$. This hierarchy and corresponding interpolation and restriction transfer operators will give rise to the algebraic multi-level system. It will be based on the structure considerations from Section 3.3.2.

3.4.1 Algebraic coarsening and transfer operators

As motivated before, we use coarse level selection techniques and prolongation/restriction operators from classical Ruge-Stüben AMG. Therefore, we give a brief review of the necessary basic facts from algebraic multigrid, cf. [TS01, Appendix A].

In AMG, a multigrid hierarchy is constructed from the structure and entries of the underlying system matrix. Geometric properties are ignored. Notation usually follows graph theory by identifying variables as graph nodes or points and non-zero non-diagonal entries in the system matrix as weighted edges between these nodes.

Let $A \in \mathbb{R}^{N\times N}$ be a given M-matrix that is a stiffness or system matrix on (the finest) level L. Each variable then corresponds to one index in an index set $\mathcal{D}_L := \{1, \ldots, N\}$. Variables on coarser levels of the (algebraic) multigrid hierarchy are collected in subsets $\mathcal{D}_l$ with $\mathcal{D}_0 \subset \mathcal{D}_1 \subset \ldots \subset \mathcal{D}_L$. Classical Ruge-Stüben AMG classifies variables on each level $0 \leqslant l < L$ into disjoint sets of coarse grid variables C^l and fine grid variables F^l, such that $\mathcal{D}_l = C^l \cup F^l$. The coarse grid variables are reused on the next coarser level, i.e. $\mathcal{D}_{l-1} := C^l$.

To formulate an algorithm for the choice of fine and coarse grid points, – we call this choice *C/F splitting* – we need some further notation. The neighborhood of a variable $i \in \mathcal{D}_l$ is given by

$$\mathcal{N}_i^l := \left\{ j \in \mathcal{D}^l \,\middle|\, j \neq i, a_{ij}^l \neq 0 \right\} ,$$

where the a_{ij}^l are the matrix entries of the (subsequently constructed) coarse grid operator on level l. A variable i is *strongly negatively coupled* to a variable j if we have the relation

$$-a_{ij}^l \geqslant \varepsilon_{str} \max_k |a_{ik}^l|$$

for a given, fixed $0 < \varepsilon_{str} < 1$. We collect these variables in

$$S_i^l := \left\{ j \in \mathcal{N}_i^l \,\middle|\, i \text{ strongly negatively coupled to } j \right\}$$

and further define ${S_i^l}^\top := \left\{ j \in \mathcal{D}_l \,\middle|\, i \in S_j^l \right\}$. Algorithm 6 then summarizes the standard coarsening or C/F splitting algorithm of Ruge-Stüben AMG, cf. [TS01, Appendix A].

Next, we consider the algebraic construction of prolongation and restriction operators. In difference to the geometric case, prolongation operators $\mathcal{P}_l^{l+1} \in \mathbb{R}^{|\mathcal{D}_{l+1}|\times|\mathcal{D}_l|}$ from algebraic

Algorithm 5 Standard coarsening algorithm

Require: level l, $\mathcal{D}_l$, S^l, ${S^l}^\top$

function AMGStandardCoarsening
  $F^l := \varnothing, C^l := \varnothing, U^l := \mathcal{D}_l$
  for $i \in U^l$ **do**
    $\lambda_i^l := \left|{S_i^l}^\top \cap U^l\right| + 2\left|{S_i^l}^\top \cap F^l\right|$
  while $\exists i$ s.th. $\lambda_i^l \neq 0$ **do**
    find $i_{\max} := \operatorname{argmax}_i \lambda_i^l$
    $C^l := C^l \cup \{i_{\max}\}$
    $U^l := U^l \backslash \{i_{\max}\}$
    for $j \in \left({S_i^l}^\top \cap U^l\right)$ **do**
      $F^l := F^l \cup \{j\}$
      $U^l := U^l \backslash \{j\}$
    for $i \in U^l$ **do**
      $\lambda_i^l := \left|{S_i^l}^\top \cap U^l\right| + 2\left|{S_i^l}^\top \cap F^l\right|$
  return C^l, F^l

multigrid are matrices by definition. As usual, we further assume $\mathcal{P}_{l+1}^l = {\mathcal{P}_l^{l+1}}^\top$, that is, restriction matrices are constructed from given prolongation matrices. Therefore, it is enough just to define prolongation, which is also called *interpolation* in AMG. Here, we have further notation

$$C_i^l := C^l \cap \mathcal{N}_i^l, \quad F_i^l := F^i \cap \mathcal{N}_i^l, \quad \tilde{C}_i^l := C^l \cap S_i^l, \quad \tilde{F}_i^l := F^l \cap S_i^l\,.$$

For the so-called *direct interpolation*, interpolation matrices are constructed as follows: Coarse grid variable are identically transferred form coarse to fine grid. However, fine grid variables $e_i^l,\ i \in \mathcal{F}^l$ have to be interpolated appropriately. This is done using the set of interpolatory variables $I_i^l := \tilde{C}_i^l$ that are all strongly connected coarse grid variables. The interpolation rule of direct interpolation is then given by

$$e_i^l = \sum_{j \in I_i^l} w_{ij}^l e_j^l, \quad w_{ij}^l = -\alpha_i^l \frac{a_{ij}^l}{a_{ii}^l}, \quad \alpha_i^l = \frac{\sum_{k \in \mathcal{N}_i^l} a_{ik}^l}{\sum_{j \in I_i^l} a_{ij}^l}\,.$$

Standard interpolation additionally considers strong connections between fine grid nodes. To this end, the original (coarse grid) operator matrix A_l is expanded to a matrix $\hat{A}_l$, such that we replace for each given fine grid variable $i \in F^l$, the variable $j \in \tilde{F}_i^l$ as

$$e_j \longrightarrow \sum_{k \in \mathcal{N}_j^l} a_{jk}^l e_k^l / a_{jj}^l\,.$$

The new set of interpolatory variables is $\hat{I}_i^l = \tilde{C}_i^l \cup (\bigcup_{j \in \tilde{F}_i^l} \tilde{C}_j^l)$. Standard interpolation is finally constructed by applying direct interpolation to the extended matrix $\hat{A}_l$.

In some cases, an additional Jacobi smoothing step is executed on the system matrix, leading to the so-called *Jacobi interpolation*, cf. [TS01, Appendix A]. Usually, *truncation* is applied to interpolation or prolongation matrices, in order to reduce the number of non-zero entries of the system matrix on the next coarser level. Truncation removes matrix entries beyond a relative threshold of ε_{tr}.

3.4.2 Algebraic multi-level system

Let us remember our intention to solve the linear system

$$A\boldsymbol{x} = \boldsymbol{b}\,, \tag{3.3}$$

with $A \in \mathbb{R}^{N\times N}$ an M-matrix and $\boldsymbol{b} \in \mathbb{R}^N$ an appropriate right-hand side. We can now use Algorithm 6 to introduce a hierarchy $\mathcal{D}_0 \subset \mathcal{D}_1 \subset \ldots \subset \mathcal{D}_L$ of variable indices, with $\mathcal{D}_L = \{1,\ldots,N\}$ the variable index set of the original linear system (3.3) and

$$\mathcal{D}_l = C^l \cup F^l, \quad \mathcal{D}_{l-1} := C^l \qquad\qquad \text{for all } l \in \{1,\ldots,L\}$$

the splitting into coarse and fine grid variables as in AMG. We furthermore introduce the same interpolation or prolongation operators $\mathcal{P}_l^{l+1} \in \mathbb{R}^{|\mathcal{D}_{l+1}|\times|\mathcal{D}_l|}$ and restriction operators $\mathcal{P}_{l+1}^l$ as in AMG. Depending on the given linear system, we might use direct interpolation, standard interpolation, Jacobi interpolation or combinations of these. We now construct an *algebraic multi-level system* using algebraic transfer matrices

$$\mathcal{S}_L := \begin{bmatrix} \mathcal{P}_L^0 \\ \mathcal{P}_L^1 \\ \vdots \\ \mathcal{P}_L^{L-1} \\ \mathcal{P}_L^L \end{bmatrix}, \quad \mathcal{S}^L := \left[\mathcal{P}_0^L \mathcal{P}_1^L \ldots \mathcal{P}_{L-1}^L \mathcal{P}_L^L\right].$$

The algebraic multi-level system is then given by

$$\mathcal{S}_L A \mathcal{S}^L \bar{\boldsymbol{x}} =: \quad \bar{A}\bar{\boldsymbol{x}} = \bar{\boldsymbol{b}} \quad := \mathcal{S}_L \boldsymbol{b}\,, \tag{3.4}$$

with

$$\bar{A} \in \mathbb{R}^{\bar{N}\times\bar{N}}, \quad \bar{\boldsymbol{x}}, \bar{\boldsymbol{b}} \in \mathbb{R}^{\bar{N}}, \quad \bar{N} = \sum_{l=0}^{L} |\mathcal{D}_l|\,.$$

As in the standard multi-level frame case, this system is not uniquely solvable. However, a non-unique solution $\bar{\boldsymbol{x}}$ can be projected back to a unique solution of the original system (3.3) by

$$\boldsymbol{x} = \mathcal{S}^L \bar{\boldsymbol{x}}_L\,.$$

3.4.3 Performance and complexity considerations

By construction, the algebraic multi-level linear system is larger than the original linear system (3.3). This increases the complexity of linear solvers that are applied to the multi-level system,

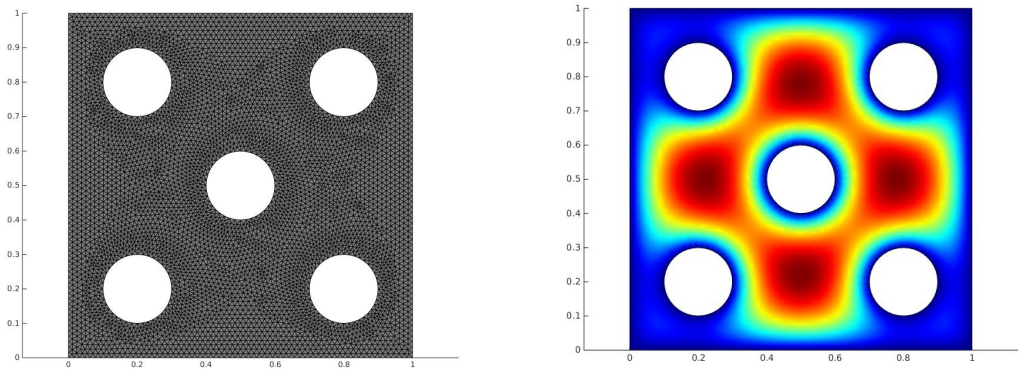


Figure 3.1: Visualization of the triangulated domain (left) and the solution field of the discussed model problem (right).

if we express the complexity in terms of the number of unknowns N of (3.3). For frame methods based on a geometric construction, it is usually possible to show that the overall complexity of the resulting iterative method stays the same, if the number of degrees of freedom of the subsequent subspaces scales like a geometric sequence. In contrast, the complexity of algebraic multigrid methods is often formalized in the so-called *operator complexity* that describes the quotient between the total number of non-zeros for all matrixes on all levels and the number of non-zeros of the system matrix on the finest level. For efficiency reasons, an operator complexity of up to two is usually expected. It is subject of future research to map the knowledge on the operator complexity in AMG to the overall complexity of the algebraic multi-level frame approach.

In real applications, optimized greedy-type solvers and improved scalability in parallelizations are expected to improve the measured runtime a lot. Therefore, even though the total complexity might be affected by the multi-level approach, a clear overall runtime improvement over standard AMG is expected for a highly optimized parallel algebraic multi-level frame solver. Such a solver is future work.

Numerical results in this article reflect a feasibility study of the proposed approach. Therefore runtime comparisons are also considered future work. Moreover, it might be possible to reduce the overall computational complexity and runtime by new on-the-fly construction and compression techniques in the future.

3.5 Numerical results

We study the solution properties of the iterative methods from Section 3.3.3 applied to the new algebraic multi-level frame construction. The model problem considered here is the Poisson

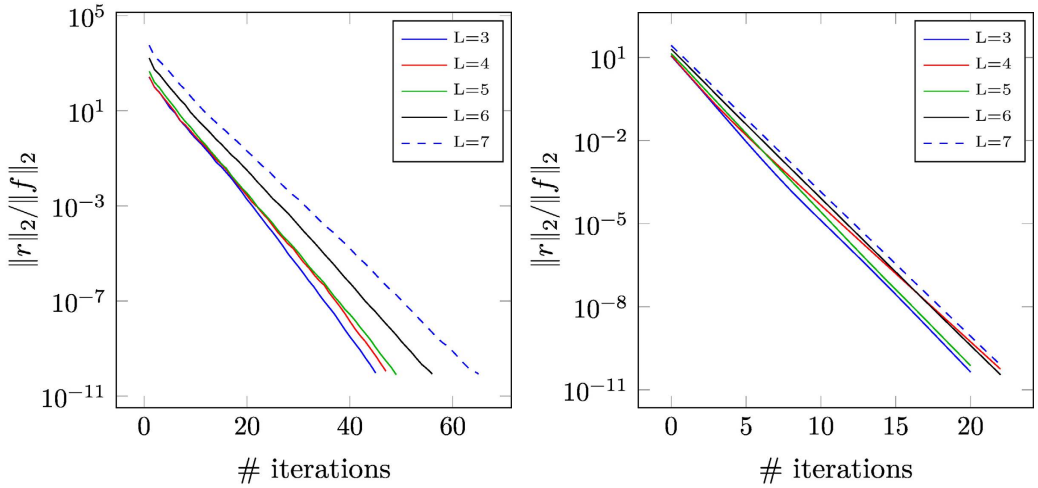


Figure 3.2: Relative residual decay for the solution of the algebraic multi-level linear system by a Jacobi-preconditioned CG method (left) and by the Gauss-Seidel method (right) for different resolution levels L.

problem

$$\begin{aligned} -\Delta u &= 1 \quad \text{in}\,\Omega\,, \\ u &= 0 \quad \text{on}\,\partial\Omega \end{aligned}$$

on the complex geometry Ω shown in Figure 3.1. It is the unit square with five circular holes.

The model problem is discretized by first-order finite elements. For our convergence studies, we define resolution levels L such that the triangulation $\mathcal{T}_L$ has a maximum triangle size of $h_{max} = 2^{-L}$. Triangulation and stiffness matrix assembly is done by COMSOL. Figure 3.1 shows on the left-hand side a triangulation for level $L = 6$. Stiffness matrices and load vectors were extracted using the COMSOL LiveLink for MATLAB extension and the command `mphmatrix`. We use *eliminated* matrices and load vectors, thus potential null spaces in the original stiffness matrix were removed by COMSOL, beforehand. The algebraic multi-level system is constructed in MATLAB.

In our MATLAB-based numerical studies, we iteratively solve the algebraic multi-level frame system up to a normalized residual norm of 10^{-9}. Normalization is done with the norm of the right-hand side load vector. The initial guess for the solution is a random vector. C/F splitting with strength parameter $\varepsilon_{str} = 0.25$ and standard interpolation with an additional Jacobi interpolation step and truncation ($\varepsilon_{tr} = 0.25$) is applied. The algebraic multigrid hierarchy has been coarsened until a maximum of 10 coarse grid variables was left.

Remember that the model problem is given on a complex geometry. Here, a standard (non-algebraic) multi-level frame construction would require a considerable effort to construct special problem-dependent coarsening, prolongations and restrictions. It might be even necessary to exactly resolve the boundary on the coarsest grid. In contrast, our algebraic approach works without any modification.

Figure 3.2 shows on the left-hand side the residual decay for an iterative solution of the

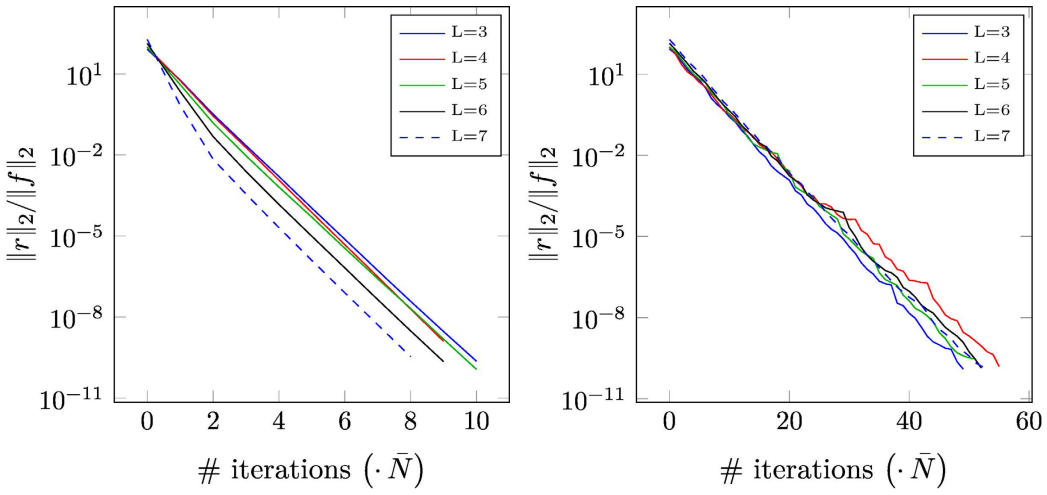


Figure 3.3: Relative residual decay for the solution of the algebraic multi-level linear system by the Gauss-Southwell method (left) and by the random Gauss-Seidel method (right) for different resolution levels L.

algebraic multi-level system by a Jacobi-preconditioned CG solver. This is equivalent to an algebraic-type BPX preconditioned solve for the standard finite element problem. A small, problem-size dependent increase in the number of iterations is visible. It is expected to stagnate for lager problems sizes. On the right-hand side of the same figure, the Gauss-Seidel iterative solver is used to solve the multi-level system. Here, optimal problem-independent convergence is shown. The method is equivalent to some algebraic multigrid method with Gauss-Seidel smoother. The Gauss-Seidel results match the results given in [GO12] for the geometric multi-level frame construction.

Results for the Gauss-Southwell iterative method, applied to the algebraic multi-level system, are given on the left-hand side of Figure 3.3. Remember that a single Gauss-Southwell step corrects a single variable. Therefore, we here denote $\bar{N}$ correction steps as a single iteration of the Gauss-Southwell method. Following [The10], one such iteration can be computed with a computational complexity of $O(\bar{N} \log \bar{N})$. Optimal problem-independent convergence is achieved. Moreover, the number of iterations is only half of the number of Gauss-Seidel iterations.

The right-hand side of Figure 3.3 gives convergence results for the random Gauss-Seidel method with the same definition of an iteration as in the Gauss-Southwell case. Here again, problem-size independent convergence is achieved. Note that this method performs corrections steps in a purely random ordering. Therefore, this approach is resilient by construction. The random Gauss-Seidel method needs about five times the amount of iterations of the Gauss-Southwell method and about 2.5 times the number of iterations of the standard Gauss-Seidel method. This relation roughly holds across all levels. All results shown in Figure 3.3 match convergence results from [GO12] in a qualitative way.

Finally, we compare the k-random block-Gauss-Seidel method for different block sizes k with the standard Gauss-Seidel method and the Gauss-Southwell method, cf. Figure 3.4. This is done for a fixed resolution level $L = 7$. Our results qualitatively match those in [GO12].

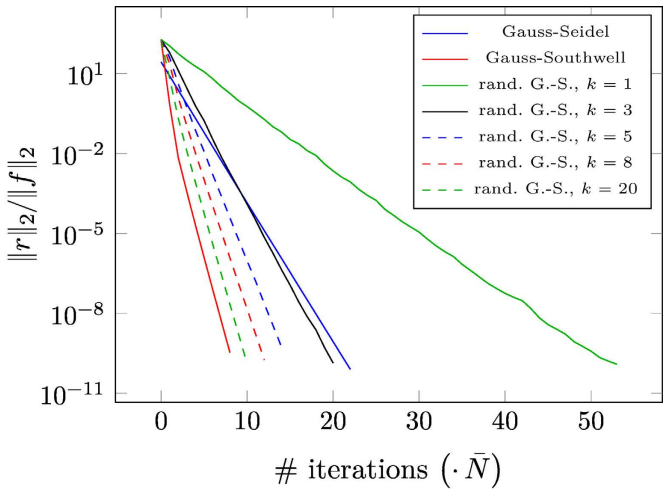


Figure 3.4: Comparison of the different relative residual decays for a complex geometry Poisson problem on level $L = 7$, solved by the Gauss-Seidel method, the Gauss-Southwell method and the k-random block-Gauss-Seidel method.

For $k \geqslant 3$, the k-random block-Gauss-Seidel method performs at least as good as the Gauss-Seidel method, i.e. an algebraic multigrid method with Gauss-Seidel smoother in the standard formulation. In case of growing k, the necessary number of iterations seems to converge towards results of the Gauss-Southwell method.

3.6 Conclusions

We have introduced an algebraic multi-level frame construction. It transforms the multi-level discretization approach, known as multi-level frames, to a purely algebraic solver technique. In case of an elliptic model problem, subspace correction iterative methods applied to the algebraic multi-level frame system converge in a problem-independent way, i.e. achieve a similar performance as standard algebraic multigrid. Moreover, the proposed approach exposes structure that will make it an optimal candidate for error-resilience. Compared to the geometric construction shown in [GO12], we achieve (qualitatively) identical results, even on a complex geometry. Due to its algebraic nature, the proposed method is an optimal candidate for generic linear algebra libraries.

In the future, it is planned to discuss improvements for performance and computational complexity. New on-the-fly construction and compression techniques may reduce the runtime and computational complexity. At the same time, the extreme parallelism and error-resilience of the proposed method is expected to be exemplified by providing a multi-level algebraic frames library based on work in [The10]. This library might to be able to outperform classical algebraic multigrid approaches on multi- and many-core architectures.

Acknowledgement

The author was partially supported by the project EXAHD of the DFG priority program 1648 Software for Exascale Computing (SPPEXA).

References

[ATDH13] H. Anzt, S. Tomov, J. Dongarra, and V. Heuveline. Weighted block-asynchronous iteration on GPU-accelerated systems. In I. Caragiannis, M. Alexander, R. Badia, M. Cannataro, A. Costan, M. Danelutto, F. Desprez, B. Krammer, J. Sahuquillo, S. Scott, and J. Weidendorfer, editors, *Euro-Par 2012: Parallel Processing Workshops*, volume 7640 of *Lecture Notes in Computer Science*, pages 145–154. Springer Berlin Heidelberg, 2013.

[ATG+12] H. Anzt, S. Tomov, M. Gates, J. Dongarra, and V. Heuveline. Block-asynchronous multigrid smoothers for GPU-accelerated systems. *Procedia Computer Science*, 9(0):7 – 16, 2012. Proceedings of the International Conference on Computational Science, ICCS 2012.

[BFG+12] A. Baker, R. Falgout, T. Gamblin, T. Kolev, M. Schulz, and U. Yang. Scaling algebraic multigrid solvers: On the road to Exascale. In C. Bischof, H.-G. Hegering, W. E. Nagel, and G. Wittum, editors, *Competence in High Performance Computing 2010*, pages 215–226. Springer Berlin Heidelberg, 2012.

[BT13] A. Beck and L. Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013.

[CdSBS12] M. Casas, B. R. de Supinski, G. Bronevetsky, and M. Schulz. Fault resilience of the algebraic multi-grid solver. In *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS '12, pages 91–100, New York, NY, USA, 2012. ACM.

[CGG+14] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward Exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1), 2014.

[Dah97] W. Dahmen. Wavelet and multiscale methods for operator equations. *Acta Numerica*, 6:55–228, 1 1997.

[DFR+04] S. Dahlke, M. Fornasier, T. Raasch, R. Stevenson, and M. Werner. Adaptive frame methods for elliptic operator equations: The steepest descent approach. *Adv. Comput. Math*, 27:27–63, 2004.

[DIR92] J. Dendy, Jr., M. Ida, and J. Rutledge. A semicoarsening multigrid algorithm for SIMD machines. *SIAM Journal on Scientific and Statistical Computing*, 13(6):1460–1469, 1992.

[FS00] A. Frommer and D. B. Szyld. On asynchronous iterations. *Journal of Computational and Applied Mathematics*, 123(1âĂŞ2):201 – 216, 2000. Numerical Analysis 2000. Vol. III: Linear Algebra.

[GHO15] M. Griebel, A. Hullmann, and P. Oswald. Optimal scaling parameters for sparse grid discretizations. *Numerical Linear Algebra with Applications*, 22(1):76–100, 2015.

[GK01] M. Griebel and F. Kiefer. Generalized hierarchical basis multigrid methods for convection-diffusion problems. SFB Preprint 720, Sonderforschungsbereich 256, Institut für Angewandte Mathematik, Universität Bonn, 2001.

[GKK03] T. Gerstner, F. Kiefer, and A. Kunoth. Wavelet and multigrid methods for convection–diffusion equations. In H.-J. Neugebauer and C. Simmer, editors, *Dynamics of Multiscale Earth Systems*, Lecture Notes in Earth Sciences 97, pages 123–134. Springer, 2003.

[GMOS06] M. Griebel, B. Metsch, D. Oeltz, and M. A. Schweitzer. Coarse grid classification: A parallel coarsening scheme for algebraic multigrid methods. *Numerical Linear Algebra with Applications*, 13(2–3):193–214, 2006.

[GO93] M. Griebel and P. Oswald. On additive Schwarz preconditioners for sparse grid discretizations. *Numerische Mathematik*, 66(1):449–463, 1993.

[GO95a] M. Griebel and P. Oswald. On the abstract theory of additive and multiplicative Schwarz algorithms. *Numerische Mathematik*, 70(2):163–180, 1995.

[GO95b] M. Griebel and P. Oswald. Tensor product type subspace splittings and multilevel iterative methods for anisotropic problems. *Advances in Computational Mathematics*, 4(1):171–206, 1995.

[GO12] M. Griebel and P. Oswald. Greedy and randomized versions of the multiplicative Schwarz method. *Linear Algebra and its Applications*, 7:1596–1610, 2012.

[Gri93] M. Griebel. *Multilevelmethoden als Iterationsverfahren über Erzeugendensystemen*. Teubner Skripten zur Numerik. Vieweg+Teubner Verlag, 1993.

[Gri94] M. Griebel. Multilevel algorithms considered as iterative methods on semidefinite systems. *SIAM Int. J. Sci. Stat. Comput.*, 15(3):547–565, 1994.

[HGRW15] M. Huber, B. Gmeiner, U. Rüde, and B. I. Wohlmuth. Resilience for Exascale enabled multigrid methods. *CoRR*, abs/1501.07400, 2015.

[HSS08] H. Harbrecht, R. Schneider, and C. Schwab. Multilevel frames for sparse tensor product spaces. *Numerische Mathematik*, 110(2):199–220, 2008.

[Kie01] F. Kiefer. *Multiskalen-Verfahren für Konvektions-Diffusions Probleme*. PhD thesis, Institute for Numerical Simulation, University of Bonn, jul. 2001.

[Mai13] J. Mairal. Optimization with first-order surrogate functions. In *Proceedings of the International Conference on Machine Learning (ICML*, 2013.

[Met13] B. Metsch. *Algebraic Multigrid (AMG) for Saddle Point Systems*. PhD thesis, Institute for Numerical Simulation, University of Bonn, jul. 2013.

[MS07] S. MacLachlan and Y. Saad. A greedy strategy for coarse-grid selection. *SIAM Journal on Scientific Computing*, 29(5):1825–1853, 2007.

[Nes12] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[OGWW98] C. Oosterlee, F. Gaspar, T. Washio, and R. Wienands. Multigrid line smoothers for higher order upwind discretizations of convection-dominated problems. *Journal of Computational Physics*, 139(2):274 – 307, 1998.

[OZ] P. Oswald and W. Zhou. Convergence estimates for Kaczmarz-type methods. *Linear Algebra and its Applications*. , 2015, submitted.

[RS86] J. Ruge and K. Stüben. Algebraic multigrid (AMG). In S. McCormick, editor, *Multigrid Methods, Frontiers in Applied Mathematics*, volume 5. SIAM, Philadelphia, 1986.

[Sou40] R. Southwell. *Relaxation methods in engineering science - A treatise on approximate computation*. Oxford Univ. Press, Oxford, 1940.

[Stü01] K. Stüben. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, 128(1-2):281 – 309, 2001. Numerical Analysis 2000. Vol. VII: Partial Differential Equations.

[The10] R. Thesen. Effiziente adaptive Lösung von Multilevelsystemen. diploma thesis, Institute for Numerical Simulation, University of Bonn, aug. 2010.

[TS01] U. Trottenberg and A. Schuller. *Multigrid*. Academic Press, Inc., Orlando, FL, USA, 2001.

[VMB96] P. Vank, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196, 1996.

[ZF13] A. Zouzias and N. M. Freris. Randomized extended Kaczmarz for solving least squares. *SIAM Journal on Matrix Analysis and Applications*, 34(2):773–793, 2013.

4 On the algebraic construction of sparse multilevel approximations of elliptic tensor product problems

4.1 Introduction

The solution of elliptic problems on tensor products of a polygonally bounded domain $\Omega \subset \mathbb{R}^d$ with e.g. $d = 2, 3$ given by

$$\begin{aligned}(\Delta \otimes \Delta)u &= f \quad \text{on } \Omega \times \Omega\,,\\ u &= 0 \quad \text{on } \partial(\Omega \times \Omega)\,,\end{aligned}$$

is an important high-dimensional problem. As an example, this problem shows up in the estimation of the output covariance of an elliptic partial differential equation with random input data that is given on a domain Ω, see [Har10, HPS13, ST03b, ST03a] for example. The problem becomes high-dimensional since the dimensionality of the elliptic problem on Ω is doubled. In case of real-world problems in $d = 3$, we end up solving a six-dimensional problem, which might become prohibitively expensive.

Recently, there have been developments to overcome this strong limitation. These developments are based on the introduction of a *geometrically constructed multilevel frame* to solve the elliptic problem on Ω. Standard Galerkin discretizations of this problem approximate the solution with respect to a *basis* of a finite-dimensional trial and test space V_J associated to a triangulation $\mathcal{T}_J$ of the domain Ω. A multi-level frame discretization uses more functions to construct the trial and test space. In fact, it uses all basis functions of a (nested) hierarchy of subspaces $V_0 \subset V_1 \subset \ldots \subset V_J$, which are associated to a (nested) *geometric* hierarchy of triangulations $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_J$ with an increasing number of nodes $|\mathcal{T}_0| < |\mathcal{T}_1| < \ldots < |\mathcal{T}_J|$. This set with many redundant basis functions is no longer a basis for V_J, but a *frame*.

The multilevel frame gives rise to a sparse approximation with respect to the interaction of the involved domains in $\Omega \times \Omega$ [HSS08, ST03b, ST03a]. It has been shown that the sparse approximation, i.e. using the trial and test space $\bigcup_{0 \leqslant j+j' \leqslant J} V_j \otimes V_{j'}$ instead of $V_J \otimes V_J$, allows to solve the tensor product problem at a computational complexity that stays essentially (i.e. up to a poly-logarithmic factor) proportional to the number of degrees of freedom to discretize a function on the single domain Ω with respect to the trial space V_J. In a more recent work by one of the authors [HPS13], it has been shown that the sparse approximation can equivalently be replaced by the sparse grid combination technique [BG04, GH14, GSZ92, HGC07], which combines cheap anisotropic full-grid solutions of the tensor-product elliptic problem. This further reduces the computational work and facilitates the implementation.

However, the currently available geometric construction of the multilevel hierarchy imposes limitations on the discretization for real-world problems. First, the coarsest triangulation $\mathcal{T}_0$ in

the geometrical hierarchy of triangulations has to fully represent the boundary of the geometry Ω. This either limits the types of geometry to consider or the computational efficiency (in case even the coarsest mesh has to be fine at the boundary). Second, the use of a fully unstructured mesh $\mathcal{T}_J$ becomes barely possible, since we are missing a coarsening strategy for such a mesh.

This work introduces *algebraically* constructed multilevel hierarchies [Gri94, GO12, Zas16] for the solution of elliptic problems on tensor product domains. While previous works [HPS13, HSS08] first constructed the multilevel hierarchy of meshes or triangulations and then discretized the problem by finite elements, the new approach first discretizes the problem on Ω on the finest (potentially unstructured) mesh $\mathcal{T}_J$ and then constructs coarser versions of the linear system resulting from the fine discretization. The coarser problems are generated using algebraic coarsening known from the classical *Ruge-Stüben algebraic multigrid* (AMG) [RS86, Stü01]. The algebraic construction of multilevel hierarchies for frames has been previously discussed in context of optimal complexity solvers for elliptic problems in [Zas16]. However, it has not been applied in the context of sparse approximation yet. Note that, by construction, our new approach allows us to overcome both the limitations in presence of complex geometries and the requirements on the structure of the mesh. Moreover, it perfectly fits into the context of black-box type PDE solvers.

As it is well-known, a full theory for algebraic multigrid methods, especially in the multilevel context and on unstructured grids, is still to be developed. Nevertheless, this technique is extremely popular as solver in real-world applications and, usually, empirically shows the same performance as geometric multigrid. This work follows the same spirit and focuses on the formal construction and the empirical analysis of the resulting numerical method. Thereby, we are able to match the convergence results available for geometrically constructed sparse approximations, while being able to apply this approach to complex geometries and unstructured grids in a black-box fashion.

In Section 4.2, the algebraic multilevel construction is outlined. This construction is introduced to the tensor product problem with sparse approximation and the sparse grid combination technique in Section 4.3. Section 4.4 briefly discusses the implementation. In Section 4.5, we give a series of numerical examples with empirical error analysis. Finally, Section 4.6 summarizes this work.

4.2 Algebraic multilevel constructions

In our algebraic construction, we aim at replacing classical multilevel discretizations for elliptic partial differential equations by a purely matrix-based construction. That is, we consider an elliptic partial differential equation

$$\begin{aligned} -\Delta u &= f \quad \text{on } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned} \tag{4.1}$$

on a polygonally bounded domain $\Omega \subset \mathbb{R}^d$. This problem has been discretized by some method on a discretization level J, leading to a system of linear equations

$$\boldsymbol{A}_J \boldsymbol{u}_J = \boldsymbol{f}_J \,, \tag{4.2}$$

Algorithm 6 Standard coarsening algorithm [TS01]

Require: level j

function AMGStandardCoarsening
  $\mathcal{F}_j := \emptyset, \mathcal{D}_{j-1} := \emptyset, \mathcal{U}_j := \mathcal{D}_j$
  for $i \in \mathcal{U}_j$ **do**
    $\lambda_j(i) := \left|\mathcal{S}_j(i)^\top \cap \mathcal{U}_j\right| + 2\left|\mathcal{S}_j(i)^\top \cap \mathcal{F}_j\right|$
  while $\exists i$ s.th. $\lambda_j(i) \neq 0$ **do**
    find $i_{\max} := \operatorname{argmax}_i \lambda_j(i)$
    $\mathcal{D}_{j-1} := \mathcal{D}_{j-1} \cup \{i_{\max}\}$
    $\mathcal{U}_j := \mathcal{U}_j \backslash \{i_{\max}\}$
    for $k \in (\mathcal{S}_j(i_{\max})^\top \cap \mathcal{U}_j)$ **do**
      $\mathcal{F}_j := \mathcal{F}_j \cup \{k\}$
      $\mathcal{U}_j := \mathcal{U}_j \backslash \{k\}$
    for $i \in U_j$ **do**
      $\lambda_j(i) := \left|\mathcal{S}_j(i)^\top \cap \mathcal{U}_j\right| + 2\left|\mathcal{S}_j(i)^\top \cap \mathcal{F}_j\right|$
  return $\mathcal{D}_{j-1}, \mathcal{F}_j$

where $\boldsymbol{A}_J \in \mathbb{R}^{N_J \times N_J}$ is an *M-matrix* and $\boldsymbol{u}_J, \boldsymbol{f}_J \in \mathbb{R}^{N_J}$. Note that an M-matrix has positive diagonal entries, non-positive non-diagonal entries, is non-singular and the entries of its inverse are non-negative. In case of the discretization by finite elements, $\boldsymbol{A}_J$ corresponds to the stiffness matrix and $\boldsymbol{f}_J$ is the load vector, obtained by, for example, using the mass matrix $\boldsymbol{M}_J$ and interpolation. Moreover, we identify each variable $u_{J,i}$ in $\boldsymbol{u}_J = (u_{J,1} \dots u_{J,N_J})^\top$ by its index i and introduce the corresponding index set $\mathcal{D}_J := \{1, \dots, N_J\}$ for discretization level J.

4.2.1 Multilevel hierarchy of discretized problems

The objective is to construct from (4.2) a hierarchy of systems of linear equations

$$\boldsymbol{A}_j \boldsymbol{u}_j = \boldsymbol{f}_j\,, \quad j = 0, \dots, J\,, \tag{4.3}$$

which are similar to discretizations on different geometric refinement levels. Especially, we intend to do this in a purely matrix-based, i.e. algebraic, way by using coarsening and transfer operators from algebraic multigrid (AMG) [Stü01]. To this end, we first introduce a construction method for a hierarchy of variable sets

$$\mathcal{D}_0 \subset \mathcal{D}_1 \subset \ldots \subset \mathcal{D}_J \tag{4.4}$$

of sizes

$$N_0 \leqslant N_1 \leqslant \ldots \leqslant N_J\,.$$

In classical Ruge-Stüben AMG [RS86, TS01], this is achieved by recursively splitting the set

of variables $\mathcal{D}_j$ on level j into a set of coarse and fine grid variables

$$\mathcal{D}_j = \mathcal{D}_{j-1} \cup \mathcal{F}_j \,,$$

where "$\cup$" is the union of two disjoint sets. Each fine grid variable is supposed to be in the neighborhood of an appropriate amount of strongly negatively coupled coarse grid variables, where we define the *neighborhood* of a variable $i \in \mathcal{D}_j$ by

$$\mathcal{N}_j(i) := \{i' \in \mathcal{D}_j : i' \neq i,\ a_{j,ii'} \neq 0\} \,,$$

where $\boldsymbol{A}_j = (a_{j,ii'})_{i,i'=1}^{N_j}$. That is, we consider neighborhoods between variables by reinterpreting the system matrix $\boldsymbol{A}_j$ as the adjacency matrix of a graph with edges between nodes for each non-zero matrix entry. Moreover, the set of neighboring strongly negatively coupled variables of a variable i is

$$\mathcal{S}_j(i) := \left\{ i' \in \mathcal{N}_j(i) : -a_{j,ii'} \geqslant \epsilon_{str} \max_k |a_{j,ik}| \right\}$$

with a strength measure $0 < \epsilon_{str} < 1$. The *standard coarsening* procedure, cf. Algorithm 6 [TS01], builds an appropriate splitting $\mathcal{D}_j = \mathcal{D}_{j-1} \cup \mathcal{F}_j$ based on these considerations. It also involves the sets $\mathcal{S}_j(i)^\top$, which are given by

$$\mathcal{S}_j(i)^\top := \{i' \in \mathcal{D}_j : i \in \mathcal{S}_j(i')\} \,.$$

Algorithm 6 uses the notation $|\cdot|$ to express the cardinality of a set.

In order to define the hierarchy of linear systems (4.3), we further need a means to transfer information between two consecutive levels j and $j+1$. This is done by prolongation operators $\boldsymbol{P}_j^{j+1} \in \mathbb{R}^{N_{j+1} \times N_j}$ and restriction operators $\boldsymbol{P}_{j+1}^{j} \in \mathbb{R}^{N_j \times N_{j+1}}$. Prolongation and restriction are done in a purely algebraic way based on AMG. In *standard interpolation* [TS01], which is one possible type of algebraic prolongation, data given on a fine grid node $i \in \mathcal{F}_j$ is interpolated from the set of interpolatory variables

$$\mathcal{I}_j(i) := (\mathcal{D}_{j-1} \cap \mathcal{S}_j(i)) \cap \left(\bigcup_{i' \in \mathcal{F}_j \cap \mathcal{S}_j(i)} (\mathcal{D}_{j-1} \cap \mathcal{S}_j(i')) \right) .$$

Thus, it is interpolated from strongly negatively coupled coarse grid points and all coarse grid points that are strongly negatively coupled to strongly negatively coupled fine grid points. The exact choice of prolongation / interpolation weights is known from literature [TS01]. If the quality of the resulting algebraic interpolation is not good enough, one might also apply one or several steps of *Jacobi interpolation* [TS01]. This, roughly speaking, extends the whole set of interpolatory variables $\mathcal{I}_j(i)$ of a node i by one layer of additional neighboring nodes. *Truncation* allows to drop some interpolatory variables based on a threshold [TS01]. *Restriction* is given as the transpose of the prolongation, i.e. $\boldsymbol{P}_{j+1}^{j} = \boldsymbol{P}_j^{j+1\top}$.

Finally, we recursively define for $j = J-1, \ldots, 0$ the matrices and the right-hand sides involved in the hierarchy of linear systems (4.3) as

$$\boldsymbol{A}_j := \boldsymbol{P}_{j+1}^{j} \boldsymbol{A}_{j+1} \boldsymbol{P}_j^{j+1}, \quad \boldsymbol{f}_j := \boldsymbol{P}_{j+1}^{j} \boldsymbol{f}_{j+1} \,,$$

Algorithm 7 V-cycle in a multigrid scheme

Require: $\boldsymbol{A}_0, \ldots, \boldsymbol{A}_J, \boldsymbol{P}_0^1, \ldots, \boldsymbol{P}_{J-1}^J, \boldsymbol{P}_J^{J-1}, \ldots, \boldsymbol{P}_1^0$

function VCycle($\boldsymbol{u}_j$, $\boldsymbol{b}_j$, j)
  if j=0 **then**
    return $\boldsymbol{A}_j^{-1}\boldsymbol{b}_j$ ▷ direct solve on coarsest level
  else
    $\boldsymbol{u}_j$ = SMOOTHER($\boldsymbol{u}_j$, $\boldsymbol{b}_j$) ▷ pre-smoothing
    $\boldsymbol{r}_{j-1} = \boldsymbol{P}_j^{j-1}(\boldsymbol{b} - \boldsymbol{A}_j\boldsymbol{u}_j)$ ▷ restriction
    $\boldsymbol{u}_{j-1}$ = VCycle($\boldsymbol{0}$,$\boldsymbol{r}_{j-1}$,$j-1$) ▷ coarse grid correction
    $\boldsymbol{u}_j = \boldsymbol{u}_j + \boldsymbol{P}_{j-1}^j\boldsymbol{u}_{j-1}$ ▷ prolongation
    $\boldsymbol{u}_j$ = SMOOTHER($\boldsymbol{u}_j$, $\boldsymbol{b}_j$) ▷ post-smoothing
    return $\boldsymbol{u}_j$

which can also be directly expressed in terms of prolongations and restrictions from $\boldsymbol{A}_J$ and $\boldsymbol{f}_J$ as

$$\boldsymbol{A}_j := \boldsymbol{P}_{j+1}^j \cdots \boldsymbol{P}_J^{J-1}\boldsymbol{A}_J\boldsymbol{P}_{J-1}^J \cdots \boldsymbol{P}_j^{j+1}, \quad \boldsymbol{f}_j := \boldsymbol{P}_{j+1}^j \cdots \boldsymbol{P}_J^{J-1}\boldsymbol{f}_J\,.$$

Later on, we will also use the abbreviations

$$\boldsymbol{P}_J^j := \boldsymbol{P}_{j+1}^j \cdots \boldsymbol{P}_J^{J-1}, \quad \boldsymbol{P}_j^J = \boldsymbol{P}_{J-1}^J \cdots \boldsymbol{P}_j^{j+1}\,. \tag{4.5}$$

Optimal complexity in AMG can be achieved, if coarser levels are constructed such that the *operator complexity*

$$C_{\boldsymbol{A}} := \sum_j \frac{\eta(\boldsymbol{A}_j)}{\eta(\boldsymbol{A}_J)}\,,$$

where $\eta(\boldsymbol{A}_J)$ is the number of non-zeros in $\boldsymbol{A}_J$, stays bounded by some constant independent of J. Standard coarsening together with standard interpolation empirically fulfill this property for model problems discretized on simple geometries. However, in more complex situations, it might happen that standard interpolation and standard coarsening fail in achieving this. Then, stronger or more aggressive versions such as *extended / multi-pass interpolation* and *aggressive coarsening* on some levels are applied to keep this empirical property [Yan10]. In fact, it might become necessary to use the operator complexity as indicator function in a manual optimization process in which several combinations of coarsenings and interpolation schemes are tried until an acceptable operator complexity is reached. Unfortunately, to the best of the authors' knowledge, there is for now no theory on the decay of the number of non-zeros in the coarse grid matrices $\boldsymbol{A}_j$ constructed by classical Ruge-Stüben AMG on multiple levels and for general M matrices $\boldsymbol{A}_J$, which could simplify this process.

In classical literature on algebraic multigrid, the hierarchy of system matrices, prolongation operators, and restriction operators

$$\boldsymbol{A}_0, \ldots, \boldsymbol{A}_J, \quad \boldsymbol{P}_0^1, \ldots, \boldsymbol{P}_{J-1}^J, \quad \boldsymbol{P}_J^{J-1}, \ldots, \boldsymbol{P}_1^0\,,$$

are used in an iterative method with, e.g., a V-cycle, cf. Algorithm 4.2.1, in order to solve the linear system (4.2) with optimal constant or logarithmically growing number of iterations.

Instead, we will use it for the construction of a multi-level hierarchy of problems as required by sparse multilevel approximations.

4.2.2 Multilevel frames

Let us note here that the above algebraic construction naturally leads to *algebraic multilevel frames*, cf. [Zas16], for the elliptic problem on Ω. That is, we formally introduce the system of linear equations

$$\boldsymbol{A_J u_J} = \boldsymbol{f_J} \tag{4.6}$$

with

$$\boldsymbol{A_J} := \begin{pmatrix} \boldsymbol{A}_{11} & \cdots & \boldsymbol{A}_{1J} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{J1} & \cdots & \boldsymbol{A}_{JJ} \end{pmatrix}, \quad \boldsymbol{u_J} := \begin{pmatrix} \boldsymbol{u}_0 \\ \vdots \\ \boldsymbol{u}_J \end{pmatrix}, \quad \boldsymbol{f_J} := \begin{pmatrix} \boldsymbol{f}_0 \\ \vdots \\ \boldsymbol{f}_J \end{pmatrix}$$

and set

$$\boldsymbol{A}_{j_1 j_2} = \boldsymbol{P}_{j_1}^J \boldsymbol{A}_J \boldsymbol{P}_J^{j_2} .$$

The diagonal matrices $\boldsymbol{A}_{jj}$ are the system matrices $\boldsymbol{A}_j$ from the previous section. Moreover, we use (4.5) to extend prolongation / restriction to arbitrary levels. We further introduce the multi-index $\boldsymbol{j} = (j_1, j_2)$ allowing the abbreviated notation

$$\boldsymbol{A_J} = [\boldsymbol{A_j}]_{\|\boldsymbol{j}\|_{\ell^\infty} \leqslant J}, \quad \boldsymbol{u_J} = [\boldsymbol{u}_j]_{|j| \leqslant J}, \quad \boldsymbol{f_J} = [\boldsymbol{f}_j]_{|j| \leqslant J}.$$

Note that matrix $\boldsymbol{A_J}$ has a large kernel. However, it can be ignored when solving (4.3.1) by using appropriate iterative linear solvers. The projection matrix

$$\mathcal{P}_{\boldsymbol{J}} = \left[\boldsymbol{P}_0^J, \boldsymbol{P}_1^J, \ldots, \boldsymbol{P}_J^J\right]$$

can be used to transfer the right-hand side $\boldsymbol{f}_J$ from (4.2) to the multi-level representation $\boldsymbol{f_J}$ and to project back solutions $\boldsymbol{u_J}$ to the single-level solutions $\boldsymbol{u}_J$. This is done by

$$\boldsymbol{u}_J = \mathcal{P}_{\boldsymbol{J}} \boldsymbol{u_J} = \sum_{j=0}^{J} \boldsymbol{P}_j^J \boldsymbol{u}_j, \qquad \boldsymbol{f_J} = \mathcal{P}_{\boldsymbol{J}}^\top \boldsymbol{f}_J = \left[\boldsymbol{P}_J^0 \boldsymbol{f}_J, \ldots, \boldsymbol{P}_J^J \boldsymbol{f}_J\right]^\top . \tag{4.7}$$

Using the linear system (4.3.1) together with the transfer operations from (4.7) instead of using linear system (4.2) conceptually corresponds to replacing a single-level discretization by a multi-level frame discretization. As in multilevel frame discretizations based on geometric refinements / coarsening, cf. [HSS08], the above system of linear equations is much larger, since it encodes the full information of the hierarchy of systems in (4.3). However, it has the big advantage that the application of standard iterative solvers such as Jacobi, Gauss-Seidel or CG to (4.3.1) immediately leads to the same convergence behavior (in terms of the number of iterations) as if these solvers were applied with a BPX-preconditioner to (4.2). A Gauss-Seidel method applied to (4.3.1) could e.g. converge as fast as a multigrid method with Gauss-Seidel smoother applied to (4.2).

From a theoretical point of view, it has been formally shown for *geometric* multi-level constructions, that (4.3.1) is equivalent to the linear system of equations (4.2), if the BPX-

preconditioner is applied in the solution process, cf. [BPX90, Dah97, Gri93, Osw94]. In [Zas16], it has been further shown by numerical experiments that the application of specific iterative solvers to (4.3.1) leads to problem-size independent convergence rates for the here discussed case of *algebraically* constructed multilevel frames.

4.3 Sparse algebraic tensor product approach

Next, we like to consider elliptic problems on tensor products $\Omega \times \Omega$ of the polygonally bounded domain Ω. That is, we consider problems of the form

$$\begin{aligned} (\Delta \otimes \Delta) u &= f \quad \text{on } \Omega \times \Omega \,, \\ u &= 0 \quad \text{on } \partial(\Omega \times \Omega) \,. \end{aligned} \tag{4.8}$$

As in Section 4.2, we assume to have a discretization (e.g. by finite elements) for the problem on a level J resulting in the system of linear equations

$$(\boldsymbol{A}_J \otimes \boldsymbol{A}_J)\boldsymbol{U}_J = \boldsymbol{F}_J \,. \tag{4.9}$$

Here, $\boldsymbol{A}_J \in \mathbb{R}^{N_J \times N_J}$ is the system matrix from (4.2). The operator $\otimes$ is the Kronecker product operator for matrices. For matrices $\boldsymbol{S} \in \mathbb{R}^{n_1 \times n_2}$, $\boldsymbol{T} \in \mathbb{R}^{m_1 \times m_2}$, it computes the Kronecker product

$$\boldsymbol{S} \otimes \boldsymbol{T} := \begin{pmatrix} s_{11}\boldsymbol{T} & \dots & s_{1n_2}\boldsymbol{T} \\ \vdots & \ddots & \vdots \\ s_{n_1 1}\boldsymbol{T} & \dots & s_{n_1 n_2}\boldsymbol{T} \end{pmatrix} .$$

Consequently, $\boldsymbol{A}_J \otimes \boldsymbol{A}_J$ becomes a matrix of size ${N_J}^2 \times {N_J}^2$. Moreover, $\boldsymbol{U}_J, \boldsymbol{F}_J \in \mathbb{R}^{N_J \cdot N_J}$ are the solution and the right-hand side, respectively.

By assuming an underlying d-dimensional finite element discretization with mesh width h and a multigrid-type linear solver, solving the linear system in (4.9) would require at least $O\left(h^{-2d}\right)$ operations, in contrast to $O\left(h^{-d}\right)$ for the problem given by (4.2). This amount of computational work is prohibitively large, especially for larger d. Therefore, we shall find a way to reduce the amount of work to solve this problem. Before we do that, we change the problem discretization to a multilevel discretization, which is the basis for the subsequent sparse approaches.

4.3.1 Multilevel frames for tensor product constructions

To extend the solution approach from Section 4.2.2 to tensor product problems, we first recall that we had in the *univariate* multilevel frame case matrix blocks of the form

$$\boldsymbol{A_j} = \boldsymbol{A}_{j_1 j_2} := \boldsymbol{P}_{j_1}^J \boldsymbol{A}_J \boldsymbol{P}_J^{j_2} \,,$$

with $\boldsymbol{P}_J^j$, $\boldsymbol{P}_j^J$ as defined in (4.5) by applying coarsening and the transfer operators of algebraic multigrid. In the univariate case, the multilevel frame linear system of equations was

$$\boldsymbol{A_J u_J} = \boldsymbol{f_J} \,,$$

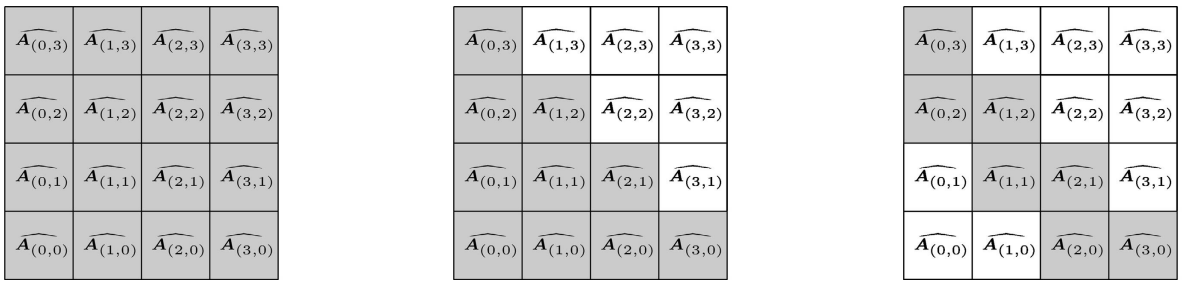


Figure 4.1: For discretization level $J = 3$, multilevel frames on the full tensor product space require a very densely populated system matrix $\widehat{\boldsymbol{A_J}}$ *(left)*, while sparse approximation leads to the system matrix $\widetilde{\boldsymbol{A_J}}$ *(center)* with smaller size due to fewer active (i.e. gray) matrix subblocks. The sparse grid combination technique *(right)* leads to the most efficient approximation.

$$\boldsymbol{A_J} = [\boldsymbol{A_j}]_{\|\boldsymbol{j}\|_{\ell^\infty} \leqslant J}, \quad \boldsymbol{u_J} = [\boldsymbol{u_j}]_{|j| \leqslant J}, \quad \boldsymbol{f_J} = [\boldsymbol{f_j}]_{|j| \leqslant J}.$$

By tensorizing this problem, we naturally get the tensor-product frame linear system of equations

$$\widehat{\boldsymbol{A_J}} \boldsymbol{U_J} = \boldsymbol{F_J}\,, \tag{4.10}$$

with

$$\widehat{\boldsymbol{A_J}} = [\boldsymbol{A}_{j_1 j_2} \otimes \boldsymbol{A}_{j_1' j_2'}]_{\|(j_1,j_1')\|_{\ell^\infty}, \|(j_2,j_2')\|_{\ell^\infty} \leqslant J}\,,$$

and

$$\boldsymbol{U_J} = [\boldsymbol{U_j}]_{\|\boldsymbol{j}\|_{\ell^\infty} \leqslant J}, \quad \boldsymbol{F_J} = [\boldsymbol{F_j}]_{\|\boldsymbol{j}\|_{\ell^\infty} \leqslant J}\,.$$

For a given right-hand side $\boldsymbol{F}_J$, we can construct the corresponding blocks $\boldsymbol{F_j}$ by

$$\boldsymbol{F_j} = \boldsymbol{F}_{j_1 j_2} := \left(\boldsymbol{P}_J^{j_1} \otimes \boldsymbol{P}_J^{j_2}\right) \boldsymbol{F}_J\,.$$

The corresponding vectors and matrices are (using $\boldsymbol{j} = (j_1, j_2)$) of the dimensionalities

$$\boldsymbol{A_j} \otimes \boldsymbol{A_{j'}} \in \mathbb{R}^{N_{j_1} N_{j_1'} \times N_{j_2} N_{j_2'}} \quad \text{and} \quad \boldsymbol{U_j}, \boldsymbol{F_j} \in \mathbb{R}^{N_{j_1} N_{j_2}}\,.$$

In order to characterize the computational complexity for the solution of (4.10), we recall that we assume to have a constant operator complexity for the sequence of matrices $\boldsymbol{A}_{jj} = \boldsymbol{A_j}$, i.e. $\sum \eta(\boldsymbol{A_j}) \leqslant c\, \eta(\boldsymbol{A}_J)$. Moreover, by definition of the Kronecker product, we have the number of non-zeros in each block of $\widehat{\boldsymbol{A_J}}$ given by

$$\eta(\boldsymbol{A_j} \otimes \boldsymbol{A_{j'}}) = \eta(\boldsymbol{A_j}) \eta(\boldsymbol{A_{j'}})\,,$$

from which it is easy to verify that we have

$$\eta(\widehat{\boldsymbol{A_J}}) = \eta(\boldsymbol{A_J}) \eta(\boldsymbol{A_J})\,,$$

with $\boldsymbol{A_J}$ from Section 4.2.2. It remains to find an upper bound to the number of non-zeros of

the univariate multilevel frame system matrix. Here, we compute

$$\eta(\boldsymbol{A}_{\boldsymbol{J}}) = \eta\left([A_{\boldsymbol{j}}]_{\|\boldsymbol{j}\|_{\ell^\infty}\leqslant J}\right) = \sum_{j_1=1}^{J}\sum_{j_2=1}^{J}\eta(\boldsymbol{A}_{j_1 j_2}) \leqslant \sum_{j_1=1}^{J}\sum_{j_2=1}^{J}\eta(\boldsymbol{A}_{\max(j_1,j_2),\max(j_1,j_2)}) = J\sum_{j}\eta(\boldsymbol{A}_{jj}) \leqslant C_{\boldsymbol{A}} J\eta(\boldsymbol{A}_J)\,.$$

In the last equality, we used that we have $\eta(\boldsymbol{A}_{j_1 j_2}) = \eta(\boldsymbol{A}_{j_2 j_1})$. The last inequality corresponds to our assumption on the operator complexity. Since we have $J \sim O(|\log h|)$, we finally get

$$\eta(\widehat{\boldsymbol{A}_{\boldsymbol{J}}}) \leqslant c\, {C_{\boldsymbol{A}}}^2 |\log h|^2 \eta(\boldsymbol{A}_J)^2\,.$$

This means that the computational work to solve (4.10) is asymptotically identical to a solve of (4.9), up to a logarithmic term. Moreover, by using recursive techniques known from the BPX-preconditioner [BPX90], we could even avoid the logarithmic term.

Figure 4.1 displays the matrix blocks $\widehat{\boldsymbol{A}_{(\boldsymbol{j},\boldsymbol{j}')}} := \boldsymbol{A}_{\max(j_1,j_2),\max(j_1',j_2')}$ that are used by the tensor product multi-level frame system. We limit ourselves to this subset of matrices for the ease of visualization. However, following [HSS08], we in fact only need these matrices to construct $\widehat{\boldsymbol{A}_J}$, if appropriate prolongation and restriction operators are considered.

4.3.2 Sparse tensor product construction

Solving (4.9) or (4.10) would be prohibitively expensive, cf. Figure 4.1. As in the geometric multilevel case, we now assume that the solution of the elliptic problem (4.1) on Ω is H^s regular. Therefore, the solution of the tensor product problem (4.8) becomes H^s_{mix}-regular, see [ST03b]. This allows to follow, for example, the lines of [HSS08] to introduce a sparse, however now algebraically constructed, version of the discretized problem. Instead of using all sub-problems for multi-indices $\|\boldsymbol{j}\|_{\ell^\infty} \leqslant J$, the sparse approximation is reduced to multi-indices $\|\boldsymbol{j}\|_{\ell^1} \leqslant J$. Thereby, we obtain a new system of linear equations

$$\widetilde{\boldsymbol{A}_{\boldsymbol{J}}}\widetilde{\boldsymbol{U}_{\boldsymbol{J}}} = \widetilde{\boldsymbol{F}_{\boldsymbol{J}}}$$

with

$$\widetilde{\boldsymbol{A}_{\boldsymbol{J}}} := [\boldsymbol{A}_{j_1 j_2} \otimes \boldsymbol{A}_{j_1' j_2'}]_{\|(j_1,j_1')\|_{\ell^1},\|(j_2,j_2')\|_{\ell^1}\leqslant J},$$

$$\widetilde{\boldsymbol{U}_{\boldsymbol{J}}} = [\boldsymbol{U}_{\boldsymbol{j}}]_{\|\boldsymbol{j}\|_{\ell^1}\leqslant J}, \quad \widetilde{\boldsymbol{F}_{\boldsymbol{J}}} = [\boldsymbol{F}_{\boldsymbol{j}}]_{\|\boldsymbol{j}\|_{\ell^1}\leqslant J}\,.$$

Figure 4.1 compares both choices in the plots on the left-hand side and the center, recalling that we use only matrices $\widehat{\boldsymbol{A}_{(\boldsymbol{j},\boldsymbol{j}')}} := \boldsymbol{A}_{\max(j_1,j_2),\max(j_1',j_2')}$ in this figure, see last section. It is easy to see, that this choice should be much more efficient.

To show that it is actually more efficient, we now discuss the number of non-zeros in $\widetilde{\boldsymbol{A}_J}$. Similar to the extimate of the number of non-zeros in the *univariate* multi-level system matrix,

we now compute

$$\begin{aligned}
\eta(\widetilde{\boldsymbol{A}_J}) &= \sum_{0\leqslant j_1+j_1'\leqslant J} \sum_{0\leqslant j_2+j_2'\leqslant J} \eta(\boldsymbol{A}_{j_1 j_2} \otimes \boldsymbol{A}_{j_1' j_2'}) \\
&= \sum_{j_1=0}^{J} \sum_{j_1'=0}^{J-j_1} \sum_{j_2=0}^{J} \sum_{j_2'=0}^{J-j_2} \eta(\boldsymbol{A}_{j_1 j_2}) \eta(\boldsymbol{A}_{j_1' j_2'}) \\
&\leqslant \sum_{j_1=0}^{J} \sum_{j_1'=0}^{J-j_1} \sum_{j_2=0}^{J} \sum_{j_2'=0}^{J-j_2} \eta(\boldsymbol{A}_{\max(j_1,j_2),\max(j_1,j_2)}) \eta(\boldsymbol{A}_{\max(j_1',j_2'),\max(j_1',j_2')}) \\
&= J^2 \sum_{j=0}^{J} \sum_{j'=0}^{J-j} \eta(\boldsymbol{A}_{jj}) \eta(\boldsymbol{A}_{j'j'})
\end{aligned}$$

As discussed before, there is not much theory on the size of the levels in the algebraic multilevel construction. The only available information is the assumed bound on the operator complexity. However, this does not give enough information to finish the above estimate. Nevertheless, the bound on the operator complexity implies a similar scaling of the non-zeros with level j as in the geometric multilevel construction. Therefore, we here assume to have the same number of non-zeros for each matrix $\boldsymbol{A}_{jj}$ as in the geometric construction, to give a hint towards the possible performance improvement by the algebraic sparse construction.

With this in mind, we follow the previous example of (linear) finite elements on a mesh with mesh width h. The number of non-zero entries for matrix $\boldsymbol{A}_j$ is proportional to the number of elements and therefore

$$\eta(\boldsymbol{A}_{jj}) = O(2^{d\,j})\,.$$

By extending the above estimate, we get

$$\begin{aligned}
\eta(\widetilde{\boldsymbol{A}_J}) = J^2 \sum_{j=0}^{J} \sum_{j'=0}^{J-j} \eta(\boldsymbol{A}_{jj}) \eta(\boldsymbol{A}_{j'j'}) &= c\,J^2 \sum_{j=0}^{J} \sum_{j'=0}^{J-j} 2^{d\,j} 2^{d\,j'} \\
= c\,J^2 \sum_{j=0}^{J} \sum_{k=j}^{J} 2^{d\,j} 2^{d\,(k-j)} = c\,J^2 \sum_{j=0}^{J} \sum_{k=j}^{J} 2^{d\,k} &= O\left(J^3 2^{d\,J}\right)
\end{aligned}$$

Moreover, we have $J = O(|\log h|)$. That is, the number of non-zeros in the system matrix in $\widetilde{\boldsymbol{A_J}}$ is asymptotically

$$\eta\left(\widetilde{\boldsymbol{A_J}}\right) = O\left(|\log h|^3\, h^{-d}\right)\,.$$

That is, in case a BPX-type preconditioner [BPX90, Dah97, Gri93, Osw94] is used, the computational complexity of the problem on the tensor product domain $\Omega \times \Omega$ is (up to a logarithmic factor) reduced to the computational complexity of the problem on domain Ω. Moreover, by applying the uni-directional principle [BZ96, Bun97, Zei11], one arrives at linear complexity for the matrix-vector product with respect to the number $\mathcal{O}(N_J \log N_J)$ of degrees of freedom in the sparse multilevel frame.

4.3.3 Sparse grid combination technique

It has been shown in [HPS13] that the previous sparse approximation is equivalent to the so-called sparse grid combination technique. The latter one starts approximating tensor product problems from a sequence of finite dimensional function spaces

$$V_0^{(i)} \subset V_1^{(i)} \subset \ldots \subset V_J^{(i)} \subset \ldots \subset V^{(i)}$$

of increasing accuracy, where i indicates the domain to which the function space is associated. Since we operate on $\Omega \times \Omega$, we have $i = 1, 2$. As next step, hierarchical increment spaces $W_j^{(i)}$ are considered such that

$$V_j^{(i)} := W_j^{(i)} \oplus V_{j-1}^{(i)} \,,$$

where $W_0^{(i)} := V_0^{(i)}$. As usual in sparse (grid) approximation, the (two-dimensional) sparse approximation space $\widehat{V_J}$ is then, cf. [GH13], defined as

$$\begin{aligned} \widehat{V_J} &:= \bigoplus_{j'=0}^{J} W_{J-j'}^{(1)} \otimes V_{j'}^{(2)} = \bigoplus_{j'=0}^{J} \left(V_{J-j'}^{(1)} \ominus V_{J-1-j'}^{(1)} \right) \otimes V_{j'}^{(2)} \\ &= \bigoplus_{j'=0}^{J} \left[\left(V_{J-j'}^{(1)} \otimes V_{j'}^{(2)} \right) \ominus \left(V_{J-1-j'}^{(1)} \otimes V_{j'}^{(2)} \right) \right] . \end{aligned} \tag{4.11}$$

The *combination technique* computes (anisotropic) full-grid solutions on the subspaces involved in equation (4.11) and combines them using appropriate projection. Translated to our problem setting, this approximation is given as

$$\begin{aligned} \widehat{\boldsymbol{U_J}} &= \sum_{j'=0}^{J} \left[\left(\boldsymbol{P}_{J-j'}^{J} \otimes \boldsymbol{P}_{j'}^{J} \right) \boldsymbol{U}_{J-j',j'} - \left(\boldsymbol{P}_{J-1-j'}^{J} \otimes \boldsymbol{P}_{j'}^{J} \right) \boldsymbol{U}_{J-1-j',j'} \right] \\ &= \sum_{\|\boldsymbol{j}\|_{\ell^1}=J} (\boldsymbol{P}_{j}^{J} \otimes \boldsymbol{P}_{j'}^{J}) \boldsymbol{U_j} \quad - \sum_{\|\boldsymbol{j}\|_{\ell^1}=J-1} (\boldsymbol{P}_{j}^{J} \otimes \boldsymbol{P}_{j'}^{J}) \boldsymbol{U_j} \,. \end{aligned} \tag{4.12}$$

To compute it, we have to solve the decoupled problems

$$\widehat{\boldsymbol{A}_{\boldsymbol{j}}} \boldsymbol{U_j} = (\boldsymbol{A}_{j_1} \otimes \boldsymbol{A}_{j_2}) \boldsymbol{U_j} = \boldsymbol{F_j}, \quad \text{where } \|\boldsymbol{j}\|_{\ell^1} \in \{J, J-1\} \,. \tag{4.13}$$

On the right-hand side of Figure 4.1, the sub-matrices $\widehat{\boldsymbol{A}_{\boldsymbol{j}}}$ used in this approximation have been marked gray. As before, one can easily verify that the total number of non-zeros of the matrices in (4.13) is asymptotically $O\left(|\log h|\, h^{-d}\right)$ for the case of linear finite elements on a tetrahedral mesh with mesh width h in d dimensions and a geometrically constructed multilevel structure. However, Figure (4.1) easily clarifies that the pre-asymptotic number of non-zeros in the matrices involved in the combination technique is much smaller than the non-zeros in the sparse approximation discussed before.

In terms of computational complexity of the combination technique, let us remind that the (approximate) solution of each sub-problem in (4.13) can be realized by an iterative linear solver with matrix-vector products. To be more specific, tensor product versions of standard

iterative solvers can be constructed, by reshaping a given iterate $\boldsymbol{U}_{\boldsymbol{j}=(j,j')} \in \mathbb{R}^{N_j \cdot N'_j}$ (and the appropriate right-hand side) to a matrix of size $N_j \times N_{j'}$. Then, the action of one step of an iterative solver for matrix $\widehat{\boldsymbol{A}_{\boldsymbol{j}}} = \boldsymbol{A}_j \otimes \boldsymbol{A}_{j'}$ is done by first applying the iterative solver step for $\boldsymbol{A}_j$ to all $N_{j'}$ columns of the reshaped matrix and by second applying the iterative solver step for $\boldsymbol{A}_{j'}$ to all N_j rows of the reshaped matrix. One easily verifies that the Kronecker product of two matrices $\boldsymbol{A}_j$, $\boldsymbol{A}_{j'}$ with $O(N_j)$, $O(N_{j'})$ non-zeros has $O(N_j N_{j'})$ non-zeros. This leads to a computational complexity of $O(N_j N_{j'})$ for a single matrix-vector product.

Next, we observe that we actually need only a problem-size independent constant number of iterations, if we choose an appropriate solver. Since we have all prolongation and restriction operators from AMG at our disposal, we can actually build a tensor product version of algebraic multigrid. The construction of a tensor-product AMG follows the idea outlined above, i.e. we apply univariate versions of AMG to the columns and rows of a reshaped iterate $\boldsymbol{U}_{\boldsymbol{j}=(j,j')}$ of size $N_j \times N_{j'}$. The tensor-product AMG gives us the property of problem-size independent convergence for each sub-problem in (4.13), i.e. we need $O(N_j N_{j'})$ operations for each sub-problem.

While we have no theory on the number of unknowns on each level of our algebraically constructed combination technique, we can still give an analogy from the geometric setting, in order to predict the overall complexity of the method. In case our algebraical construction would behave exactly as a geometrically constructed multilevel hierarchy, we would have the relation $N_j = O(2^{dj})$. Thereby, the solution of each sub-problem would require $O(2^{d(j+j')})$ operations. Since it holds $\|\boldsymbol{j}\|_{\ell^1} \in \{J, J-1\}$, we can compute

$$\begin{aligned}\sum_{\|\boldsymbol{j}\|_{\ell^1} \in \{J, J-1\}} 2^{d(j+j')} &= \sum_{\|\boldsymbol{j}\|_{\ell^1} = J} 2^{d(j+j')} + \sum_{\|\boldsymbol{j}\|_{\ell^1} = J-1} 2^{d(j+j')} \\ &= (J+1)2^{dJ} + J2^{d(J-1)}\,.\end{aligned}$$

Hence, we would finally end up with a computational complexity of $O(J2^{dJ})$ or $O(N_J \log N_J)$.

4.4 Implementation

In our numerical results, we approximate solutions for tensor product finite element discretizations of elliptic problems based on the combination technique with $\Omega \subset \mathbb{R}^{2,3}$. To this end, we assemble system matrices for a given problem, construct the multilevel hierarchies, solve the decoupled, anisotropic problems in (4.13) and combine the solutions following the combination rule (4.12).

Assembly of system matrices. The discretization by the finite element method is done with the *Matlab PDE Toolbox* of *Matlab 2017a*. We use linear finite elements and construct meshes with maximum element size `Hmax`$= 2^{-J}$. Furthermore, we use the option `Jiggle` to optimize the mesh in quality. The stiffness matrix (incorporating boundary conditions) is constructed by using the *Matlab* command `assembleFEMatrices` with option `nullspace`. In a similar way, we extract the mass matrix. Afterwards, both matrices and the mesh node coordinates are stored to files.

Construction of the multilevel hierarchy. From within *Matlab* we call an in-house ad-hoc code that uses the parallel linear solver library *hypre* [FY02] in version 2.11.1. This library contains the implementation *BoomerAMG* of classical Ruge-Stüben AMG. The code reads the stiffness matrix from file and creates the AMG multilevel hierarchy by using *hypre*. In addition to *standard coarsening* with strength measure $\epsilon_{str} = 0.25$ and *standard interpolation*, we use two passes of *Jacobi* interpolation with a truncation of the Jacobi interpolation with a threshold of 0.001 for the two-dimensional problems and 0.01 for the three-dimensional problem (being treated in Section 4.5). All other parameters are kept as the defaults of *BoomerAMG*. After having created the multigrid hierarchy, the program stores the prolongation matrices of all created levels to files. These are read by *Matlab*.

Solution of the anisotropic tensor product problems. Based on the prolongation matrices and the system matrix $\boldsymbol{A}_J$ on the finest levels, the decoupled problems in (4.13) can be set up. As discussed before, a tensor product version of AMG is used to solve the systems of linear equations. In our implementation, we construct the sub-problem operators in (4.13) by individually multiplying the transfer operators between two consecutive levels.

Our tensor product AMG is iterated until the convergence criterion

$$\|\boldsymbol{R}_{\boldsymbol{j}}^{it}\|_{\ell^2} / \|\boldsymbol{F}_{\boldsymbol{j}}\|_{\ell^2} \leqslant \epsilon_{tol}$$

is fulfilled, where $\boldsymbol{R}_{\boldsymbol{j}}^{it}$ is the residual of the current iterate $\boldsymbol{U}_{\boldsymbol{j}}^{it}$ in the solver. Since the problems in (4.13) completely decouple, we can easily parallelize their solution process by a `parfor` loop in *Matlab*. In case an individual problem becomes very expensive, we further implemented a distributed memory parallelization for the tensor product AMG based on *Matlab*'s `distributed` function. Thereby, we overcome the limitation of a non-existing multi-core parallelization for sparse matrix-vector products in *Matlab*.

Combination of the solutions. In the combination phase, we avoid to prolongate the full partial solutions to the finest level J. Instead, we randomly chose N_{eval} nodes on the product of the finest meshes on $\Omega \times \Omega$. On these points, we evaluate the combination formula (4.12) and compute the empirical error measure

$$e(\boldsymbol{U}_{approx}) = \|\boldsymbol{U}_{approx} - \boldsymbol{U}_{ref}\|_{\ell^2} / \|\boldsymbol{U}_{ref}\|_{\ell^2},$$

where $\boldsymbol{U}_{approx}$ is the approximated solution and $\boldsymbol{U}_{ref}$ is an appropriately evaluated reference solution. Note that we do not multiply the tensor product of the prolongation with the solution. Instead, we follow the ideas from Section 4.3.3 for the construction of the tensor product AMG and apply the prolongations direction-wise. The prolongation for each sub-problem is also parallelized by a `parfor` loop.

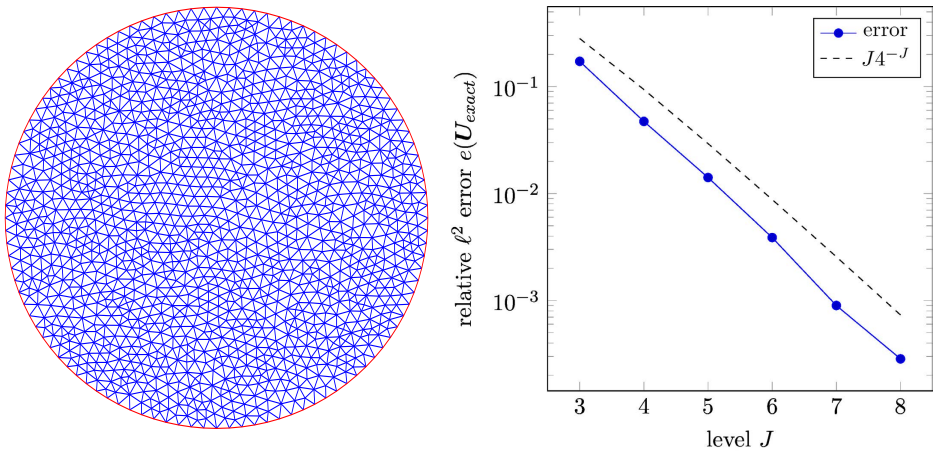


Figure 4.2: The combination technique based on our algebraic multilevel hierarchy and applied to the tensor product of a disk geometry with an unstructured mesh (*left, triangulated with* $J = 5$) shows the same convergence as the geometrically constructed combination technique *(right)*.

4.5 Numerical results

In our empirical studies, we consider the numerical solution of the problem

$$\begin{aligned} (\Delta \otimes \Delta)u &= f \quad \text{on } \Omega \times \Omega\,, \\ u &= 0 \quad \text{on } \partial(\Omega \times \Omega)\,. \end{aligned} \tag{4.14}$$

by means of the combination technique based on the algebraic multilevel hierarchy. Different choices will be made for the domain Ω and the right-hand side f.

4.5.1 Analytic example on a disk

The first study is done on a disk domain Ω with center $(0,0)^\top$ and radius 0.5. We set

$$f(\boldsymbol{x}, \boldsymbol{y}) = 1\,.$$

The exact solution of the resulting problem is

$$u(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{16}\left(x_1^2 + x_2^2 - 0.5^2\right)\left(y_1^2 + y_2^2 - 0.5^2\right)\,.$$

To approximate the solution u by the combination technique, we follow the methodology discussed in Section 4.4. As part of this, we triangulate the geometry with a maximum element width of 2^{-J}. Figure 4.2 shows on the left-hand side the resulting mesh for $J = 5$. It is obvious that the resulting mesh is unstructured. Therefore, classical geometric constructions for the sparse grid combination technique would not be feasible on that mesh. In contrast, our new algebraic approach can solve this problem.

This is shown on the right-hand side of Figure 4.2, where we compare the numerically ap-

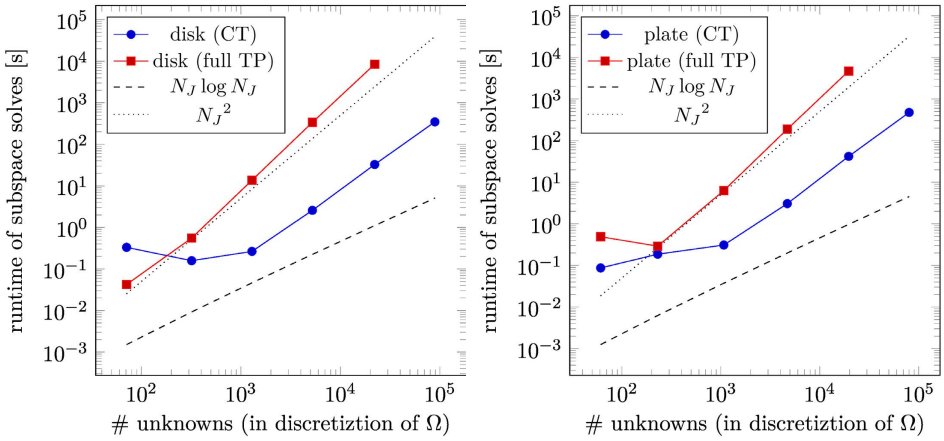


Figure 4.3: We compare the runtime of the new combination technique approach (CT) with the runtime of the traditional the full tensor-product approach (full TP) for the solution of the tensor product elliptic problems on the disk geometry *(left)* and the plate geometry *(right)*.

proximated solution against the above exact solution. Convergence results for the choices $J = 3, \ldots, 8$ are given. From literature, compare e.g. [HPS13], we know that the error of the geometrically constructed sparse grid combination technique scales for the problem under consideration like $J4^{-J}$. As we can see from the convergence results in Figure 4.2, the algebraically constructed combination technique shows the same convergence behavior, while being applicable to unstructured grids.

Figure 4.3 shows on the left-hand side computing times for growing problem size N_J of the univariate discretization of Ω. We compare the time required for the solution of the combination technique sub-problems with the time required to solve the full tensor-product problem (4.9) by our tensor-product AMG implementation. Note that we use the coarse grid hierarchies reported in Table 4.1 for both the combination technique and the full tensor-product approach. All measurements were done on a compute server with dual 20-core Intel Xeon E5-2698 v4 CPU at 2.2 GHz and 768 GB RAM. It becomes evident that our algebraically constructed combination technique approach beats the full tensor-product approach in both, computational complexity and effective runtime. However, both results do not show the predicted computational complexity of $O(N_J \log N_J)$ and $O({N_J}^2)$. There are several reasons for this behavior.

- First, algebraic multigrid often shows a small, roughly logarithmic, growth in the number of iterations for larger problem sizes, resulting in a slow-down by a logarithmic factor.
- Second, we observe a certain fill-in in the system matrices for coarser problems in the algebraic construction due to our choice of an additional (truncated) Jacobi interpolation. However, this should be pre-asymptotic behavior.
- Third, as can be seen in Table 4.1, the AMG coarsening approach chosen in our implementation does not show the exact same (asymptotic) decay rate $O(2^{dj})$ in the number

		# dofs on algebraically coarsened level j									
Ω	$J \setminus j$	0	1	2	3	4	5	6	7	8	9
disk	3	3	11	28	71						
	4	8	21	52	119	320					
	5	12	31	84	207	495	1292				
	6	20	51	139	348	852	2009	5234			
	7	35	93	244	606	1473	3510	8415	22118		
	8	46	130	366	978	2469	5983	14480	34081	89097	
	$O(2^{dj})$	1	5	22	87	348	1392	5569	22274	89097	
plate	3	5	20	61							
	4	16	36	90	230						
	5	28	68	168	414	1072					
	6	46	116	297	745	1813	4703				
	7	63	184	515	1272	3117	7491	19611			
	8	103	302	815	2124	5301	12822	30639	80146		
spanner	3	4	10	19	50	117	247				
	4	11	22	59	147	326	689	1454			
	5	40	114	300	689	1516	3216	6484	13939		
	6	210	548	1364	3123	6708	14109	29103	57438	125223	
	7	1386	3120	6627	14016	29533	61150	124921	253291	496614	1082581

Table 4.1: For a given problem on level J, the algebraic multilevel construction on our example domains Ω constructs coarser levels with a decrease of the number of unknowns roughly similar to geometric multilevel constructions e.g. in the *disk* test case. Above, only those levels j are reported that are used in the convergence study.

of levels as we expect it from the geometric construction. In fact, this leads to a problem-size dependent growth of the coarsest grid. While this growth does not affect the error decay, it shows up in the computational complexity.

Meanwhile, as stated before, we are able to beat the solution approach based on the full tensor-product approach in terms of computational complexity. Even more, if we would use AMG as solver for the anisotropic sub-problems in the *geometric* construction, we would see similar results, anyway. Finally, in terms of runtime, we are by more than two orders of magnitude faster.

4.5.2 Example on complex geometry with covariance load

The next numerical study is concerned with the solution of the problem (4.14) with the load

$$f(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(\frac{-\|\boldsymbol{x} - \boldsymbol{y}\|^2}{\ell}\right)$$

that corresponds to an (unscaled) Gaussian covariance kernel with correlation length ℓ. This is a prototype version of the tensor product elliptic problem on $\Omega \times \Omega$ showing up in the computation of the output covariance of an elliptic problem on Ω with random input, cf. [HSS08].

In addition to the more complicated right-hand side, we solve the problem for a rather complex geometry Ω. We choose the geomery of a square plate on $[0, 1]^2$ with circular wholes

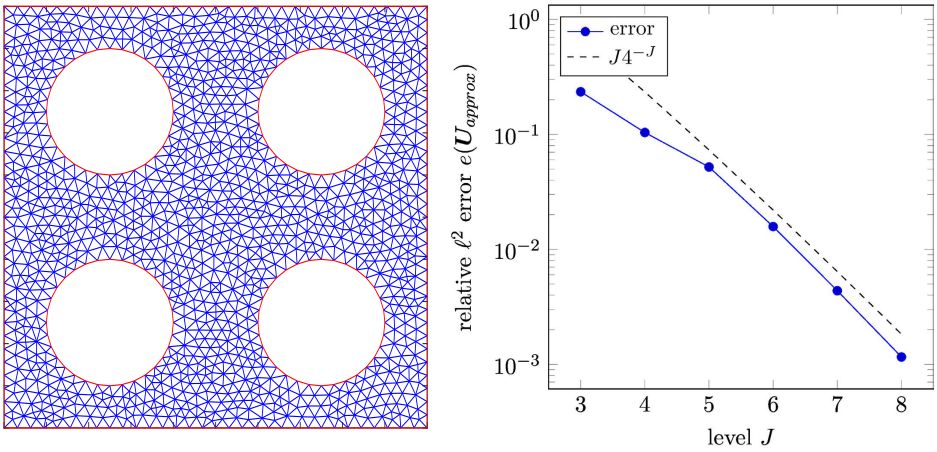


Figure 4.4: Even for a covariance load on a complex geometry (*left*, triangulated for $J = 5$), the algebraic construction shows the appropriate convergence rate after a short pre-asymptotic phase *(right)*.

of radius 0.15 which are centered at the points

$$\{(0.25, 0.25), (0.25, 0.75), (0.75, 0.25), (0.75, 0.75)\}\,.$$

Figure 4.4 shows its triangulation for $J = 5$ on the left-hand side. Note that it would be almost impossible to solve a problem on such a geometry with the geometrical construction for the sparse grid combination technique. However, with the algebraic construction, a coarsening to very few degrees of freedom becomes easily possible, compare Table 4.1.

To be able to compare the above problem against a numerically computed reference solution, we replace the (sampled) covariance kernel for $\ell = 1$ by its low-rank approximation computed with the pivoted Cholesky factorization [HPS12], truncated for a trace norm of 10^{-8}. In this case, depending on the problem size, the truncation results in roughly twenty low-rank terms.

On the right-hand side of Figure 4.4, we show the convergence results with errors computed against the numerically approximated exact solution by use of the low-rank approximation. After a pre-asymptotic phase, we are able to attain an error that scales like $J4^{-J}$ as in the geometric construction.

The problem size dependent runtime to compute the subspace solutions for the plate geometry is given in Figure 4.3 on the right-hand side. We observe similar compuational complexities and similar runtimes as in the previous example on the disk.

4.5.3 Large-scale real-world example

Our last numerical study treats a large-scale problem with a complex real-world geometry Ω. We again aim at solving (4.14) for $f(\boldsymbol{x}, \boldsymbol{y}) = 1$. However, we choose the *three-dimensional* spanner geometry found in Figure 4.5. In contrast to the previous examples, we set the maximum mesh width to 2^{5-J}, since the geometry is contained in the rather large bounding box $[-5, 5] \times [-12.2, 112] \times [-15.7, 15.7]$. Note that the triangulation of Ω results for level $J = 7$

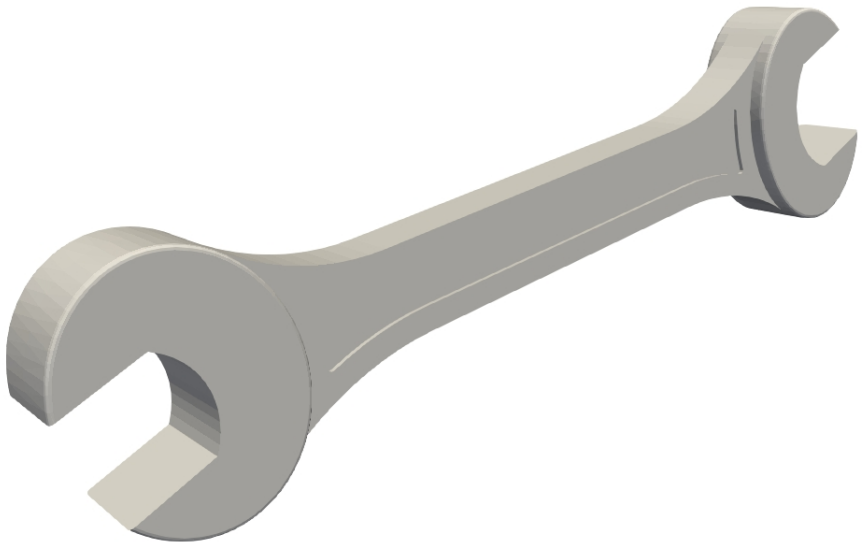


Figure 4.5: In our large-scale real world example, we solve an elliptic problem on the tensor product of the three-dimensional geometry of a spanner. For a discretization level of $J = 7$, the discretization of Ω has more than a million unknowns. This would lead to 10^{12}, that is a *trillion*, unknowns in the full tensor product discretization.

in a discretization with $1,082,581$ unknowns. That is, if we would want to solve the full tensor product problem on $\Omega \times \Omega$, cf. (4.8), then we would have to solve a problem with about 10^{12}, that is a *trilion*, unknowns. This would be clearly out of scope even for large parallel clusters. In contrast, the combination technique allows to solve this problem. Nevertheless, we still have to solve, e.g. for level $J = 7$ and the system matrix $\widehat{\boldsymbol{A}_{(0,J)}}$ a problem with $1,082,581 \times 1,386$ unknowns, compare Table 4.1.

In Figure 4.6, we show the convergence results for this large scale problem relative to a numerical approximation of the solution. Due to the high dimensionality and complexity of the domain Ω, the convergence results in Figure 4.6 are only gradually approaching the optimal scaling of $J2^{-dJ}$. Nevertheless, we are able to solve this problem up to a certain accuracy. This shows that even very complex problems of large scale can be solved by the proposed approach.

4.6 Conclusions

In this work, we have introduced an algebraic construction method for the sparse approximation of tensor product elliptic problems by means of the combination technique. While previous approaches were tight to geometric hierarchies of mesh refinements to build the underlying multilevel discretization, we were able to solve the given type of problems on complex geometries and for unstructured grids by an algebraic multilevel hierarchy based on AMG. We could show that our approach has the same convergence rates as the geometric construction. Measurements of the computational complexity were in the linear range with poly-logarithmic factors. Overall, we are now able to apply sparse approximation for elliptic tensor product problems in a black-box fashion.

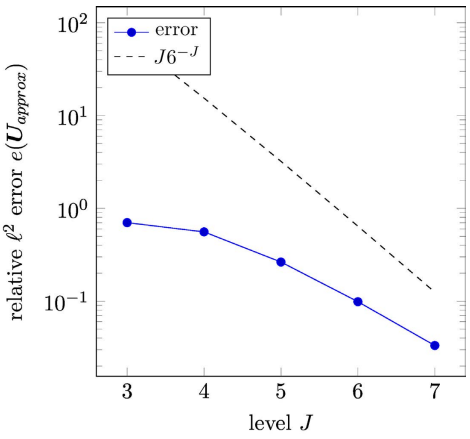


Figure 4.6: Our algebraic multilevel construction for the sparse grid combination technique on the large-scale three-dimensional spanner geometry gradually approaches the optimal convergence rate of $J2^{-dJ}$.

References

[BG04] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:1–123, 2004.

[BPX90] J. Bramble, J. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Mathematics of Computation*, 55:1–22, 1990.

[Bun97] H.-J. Bungartz. A multigrid algorithm for higher order finite elements on sparse grids. *ETNA. Electronic Transactions on Numerical Analysis*, 6:63–77, 1997.

[BZ96] R. Balder and C. Zenger. The solution of multidimensional real Helmholtz equations on sparse grids. *SIAM Journal on Scientific Computing*, 17(3):631–646, 1996.

[Dah97] W. Dahmen. Wavelet and multiscale methods for operator equations. *Acta Numerica*, 6:55–228, 1997.

[FY02] R. D. Falgout and U. M. Yang. hypre: A library of high performance preconditioners. In P. M. A. Sloot, A. G. Hoekstra, C. J. K. Tan, and J. J. Dongarra, editors, *Computational Science — ICCS 2002*, pages 632–641, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[GH13] M. Griebel and H. Harbrecht. On the construction of sparse tensor product spaces. *Mathematics of Computation*, 82(282):975–994, 2013.

[GH14] M. Griebel and H. Harbrecht. On the convergence of the combination technique. In J. Garcke and D. Pflüger, editors, *Sparse grids and Applications – Stuttgart 2014*, volume 97 of *Lecture Notes in Computational Science and Engineering*, pages 55–74. Springer, 2014.

[GO12] M. Griebel and P. Oswald. Greedy and randomized versions of the multiplicative Schwarz method. *Linear Algebra and its Applications*, 7:1596–1610, 2012.

[Gri93] M. Griebel. *Multilevelmethoden als Iterationsverfahren über Erzeugendensystemen.* Teubner Skripten zur Numerik. B.G. Teubner, Stuttgart, 1993.

[Gri94] M. Griebel. Multilevel algorithms considered as iterative methods on semidefinite systems. *SIAM International Journal Scientific Statistical Computing*, 15(3):547–565, 1994.

[GSZ92] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In P. de Groen and R. Beauwens, editors, *Iterative Methods in Linear Algebra*, pages 263–281. IMACS, Elsevier, North Holland, 1992.

[Har10] H. Harbrecht. A finite element method for elliptic problems with stochastic input data. *Applied Numerical Mathematics*, 60(3):227–244, March 2010.

[HGC07] M. Hegland, J. Garcke, and V. Challis. The combination technique and some generalisations. *Linear Algebra and its Applications*, 420(2):249–275, 2007.

[HPS12] H. Harbrecht, M. Peters, and R. Schneider. On the low-rank approximation by the pivoted Cholesky decomposition. *Applied Numerical Mathematics*, 62(4):428–440, 2012.

[HPS13] H. Harbrecht, M. Peters, and M. Siebenmorgen. Combination technique based k-th moment analysis of elliptic problems with random diffusion. *Journal of Computational Physics*, 252(C):128–141, November 2013.

[HSS08] H. Harbrecht, R. Schneider, and C. Schwab. Multilevel frames for sparse tensor product spaces. *Numerische Mathematik*, 110(2):199–220, July 2008.

[Osw94] P. Oswald. *Multilevel finite element approximation. Theory and applications.* Teubner Skripten zur Numerik. B.G. Teubner, Stuttgart, 1994.

[RS86] J. Ruge and K. Stüben. Algebraic multigrid (AMG). In S. McCormick, editor, *Multigrid Methods, Frontiers in Applied Mathematics*, volume 5. SIAM, Philadelphia, 1986.

[ST03a] C. Schwab and R. A. Todor. Sparse finite elements for stochastic elliptic problems: Higher order moments. *Computing*, 71(1):43–63, September 2003.

[ST03b] C. Schwab and R. A. Todor. Sparse finite elements for elliptic problems with stochastic loading. *Numerische Mathematik*, 95(4):707–734, 2003.

[Stü01] K. Stüben. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, 128(1-2):281–309, 2001. Numerical Analysis 2000. Vol. VII: Partial Differential Equations.

[TS01] U. Trottenberg and A. Schuller. *Multigrid.* Academic Press, Inc., Orlando, FL, USA, 2001.

[Yan10] U. M. Yang. On long-range interpolation operators for aggressive coarsening. *Numerical Linear Algebra with Applications*, 17(2-3):453–472, 2010.

[Zas16] P. Zaspel. Subspace correction methods in algebraic multi-level frames. *Linear Algebra and its Applications*, 488:505–521, 2016.

[Zei11] A. Zeiser. Fast matrix-vector multiplication in the sparse-grid Galerkin method. *SIAM Journal of Scientific Computing*, 47(3):328–346, 2011.

Part II

Contributions in context of machine learning

5 Boosting quantum machine learning models with multi-level combination technique: Pople diagrams revisited

5.1 Introduction

Chemical compound space, the property space spanned by all possible chemical compounds, is unfathomably large due to its combinatorial nature [KE04, Mul17]. Exploring chemical space from first principles is desirable in the context of computational materials design [Ced98, HWCE06, vL14] as well as to fundamentally deepen our understanding of chemistry [vL13]. Over the last couple of years overwhelming evidence has been collected indicating that quantum machine learning (QML) models, trained throughout chemical space, hold great promise to dramatically reduce the cost for predicting quantum properties, such as atomization energies of molecules, for arbitrary out-of-sample molecules [RTMvL12b, PWJ$^+$13, MRG$^+$13a, SAC$^+$17, PKLAG15, FLvLA16, DBCC16, CTS$^+$17, FHH$^+$17, HvL17a, BDP$^+$17a, GSR$^+$17, SSK$^+$18, FCHvL18]. The core idea of QML is to learn the implicit mapping from geometrical and compositional information encoded in nuclear charges and positions to corresponding electronic properties from a set of training molecules with precomputed properties at a specific level of theory. The knowledge thus obtained from training is then applied to molecules out-of-sample, i.e., molecules not in the training set. Nowadays, QML is a well-established technique and has several supervised learning variants, including mainly neural network [Beh11, SAC$^+$17, PKLAG15] and kernel ridge regression [RTMvL12b, BKC13, BPKC10]. Currently, most of the efforts towards QML in literature are devoted to developing more efficient molecular representations [HvL16, HvL17a, FCHvL18] and adapting machine learning models to a growing number of applications [JK17, GBM17, PKLAG15]. Recent overviews on the field were published in Refs. [RvL17, vL18, HSL18] and an entire issue in J. Chem. Phys. was recently devoted to the theme of "Data-enabled theoretical chemistry"[RvLB18].

This progress was made possible due to the advent of modern computers which enabled routine calculations of electronic properties such as ground state energies for large training sets of medium-sized organic molecules [RDRvL14a, SIR17, GVMVDS18] using common density functional approximations [KH15, CMSY12]. While QML prediction errors have been converged to values smaller than DFT accuracy [FHH$^+$17], the predictive power of any QML model inherently hinges on the accuracy of the employed reference data used for training. However, while the latest machine learning models are now able to make rather accurate and yet efficient predictions, the time required to compute training samples for large datasets with chemical accuracy is still prohibitive. More specifically, in order to routinely match the experimental uncertainty of thermochemistry, the highly coveted “chemical accuracy” of $\sim$1 kcal/mol, typical approximations made within density functional theory do not suffice, and computationally

expensive theories, e.g., CCSD(T) in a large basis, have to be used even when dealing just with closed-shell molecules in relaxed geometries. Unfortunately, due to its substantially larger computational complexity, the routine generation of CCSD(T) numbers in large basis sets for thousands of training molecules remains prohibitive.

The hierarchies encoded in model chemistries, well established in quantum chemistry, can be used to exploit systematic trends in cancellation of errors among different levels of theory, as proposed and demonstrated by Pople and co-workers [Pop99, HJO00]. Composite methods are based on these ideas [Kar16a], and include, among many others, Gaussian-n theories [CRTP91a, CRR$^+$99, CRRP00], the Weizmann-n methods [MO99, BOA$^+$04], and complete basis set (CBS) methods [OPMJ96, MJFOP99, MJFOP00]. They can reach chemical accuracy at the computational cost of combinations of more efficient models. When it comes to chemical space, the Pople diagram is a two-dimensional display of the relationship of the size of *any* molecule and level of theory [Pop65]. Pople diagrams can easily be extended to accommodate additional or other dimensions such as relativistic effects [KCY$^+$15] or accuracy [Kar90]. In this study, we apply the idea of a Pople diagram to combine varying levels of theory in the training set of QML models (See Fig. 1 for the general idea). More specifically, we apply the sparse-grid combination (C) technique to estimate the optimal balance among (i) electron correlation (HF, MP2, CCSD(T)), (ii) basis set size (sto-3g, 6-31g, cc-pvdz), and (iii) number of organic molecules. We find that the resulting CQML models require substantially less training instances at the computationally most demanding target level of theory.

To showcase our new developments, we will discuss a series of multi-level and multi-space machine learning models, as well as results for molecules from the QM7b dataset [MRG$^+$13b]. Using several levels in the space of electron correlation approximations already leads to a very strong improvement in the learning results, with respect to the amount of necessary training data at target accuracy. Further improvement is found by adding different levels of basis sets.

This paper is structured as follows: Section 5.3 briefly introduces the CQML model, as well as the data sets used for training and testing. In Section 5.4, results of the CQML model are presented and discussed for 2D and 3D CQML models. Finally, Section 5.5 summarizes the main-findings, draws general conclusions and presents an outlook. Appendix Section 5.6 provides detailed methodological information to facilitate reproducibility of our findings.

5.2 Computational Details

5.2.1 Datasets

Two datasets were used for proof of principle: QM7b[MRG$^+$13b] and 6k constitutional isomers[RDRvL14b] (dubbed 'CI9'), both are subsets of GDB-17 universe[RvDBR12, FBR05]. QM7b is composed of molecules with up to 7 heavy atoms, including C, N, O, S and Cl (H not counted), totaling 7211 molecules. Molecules in CI9 correspond to 6095 constitutional isomers of $C_7H_{10}O_2$.

For QM7b molecules, geometries were first optimized at the level of B3LYP/6-31g(d) using Gaussian 09 [FTS$^+$], then single point calculations were calculated using three levels of theory (HF, MP2, CCSD(T)) and three basis sets (sto-3g, 6-31g and cc-pvdz) using Molpro [WKK$^+$15], resulting in 9 single point energies per molecule.

For the CI9 molecules, three different methods were used: PM7, B3LYP/6-31g(2df,p) and G4MP2. Relaxed geometries and energies were retrieved directly from reference [RDRvL14b] for the latter two methods, while PM7 relaxed geometries and energies were obtained using MOPAC2016. [MOP]

5.2.2 QML details

We used both, the sorted Coulomb matrix [RTMvL12a, HMB$^+$13] and SLATM [HvL17a] for modeling the CI9 data set, while SLATM [HvL17a] only was used for QM7b. Though slightly better performing representations have been published previously, such as SOAP [BDP$^+$17b, WMC18], aSLATM [HvL17a] or FCHL [FCHvL18], comparison between CM and SLATM results indicates that trends are stable and that the conclusions drawn are independent of choice of representation. As kernel-functions, we have always chosen the Laplace kernel $e^{\frac{-\|\boldsymbol{R}_q-\boldsymbol{R}_i\|_1}{\sigma}}$ with σ being a hyper-parameter. The hyper-parameter σ was optimized manually and converged to $\sigma = 400$. Furthermore we use a Lavrentiev regularization of size 10^{-10}. All presented errors are mean absolute error (MAE) comparing the prediction by the CQML method with the *true* solution of the target theory level. The MAE is computed as out-of-sample error over 200 randomly chosen molecules that are not part of the training data set. These results are averaged over 20 training runs. Note that we randomly choose the $N_{\ell_{\mathrm{M}}=0}$ training molecules on the lowest level, while randomly selecting subsets of them on higher levels. This sequence of drawing ensures the nestedness of all the training samples.

5.3 Theory

In this section, we start by reviewing systematic error cancellation, composite methods, the CQML approach, kernel ridge regression based QML and Δ-ML [RDRvL15], as well as two-, and d-dimensional CQML.

5.3.1 From Pople diagrams to CQML

Telescoping series, as a means to systematic convergence of error cancellation, are a well established mathematical tool. In short, if a_n is a sequence of numbers, then

$$\sum_{\ell=1}^{N} (a_\ell - a_{\ell-1}) = a_N - a_0, \tag{5.1}$$

and if we define $\Delta_{\ell-1}^{\ell} = a_\ell - a_{\ell-1}$ and $a_0 = 0$, one has

$$a_N = a_0 + \sum_{\ell=1}^{N} \Delta_{\ell-1}^{\ell}. \tag{5.2}$$

Error cancellation is also at the root of many common practices in theoretical chemistry. Most notable are composite methods [CRTP91a, CRR$^+$99, CRRP00, MO99, BOA$^+$04, OPMJ96, MJFOP99, MJFOP00, GHH11, GHH14, CGH18], recently reviewed in Ref. [Kar16b], which

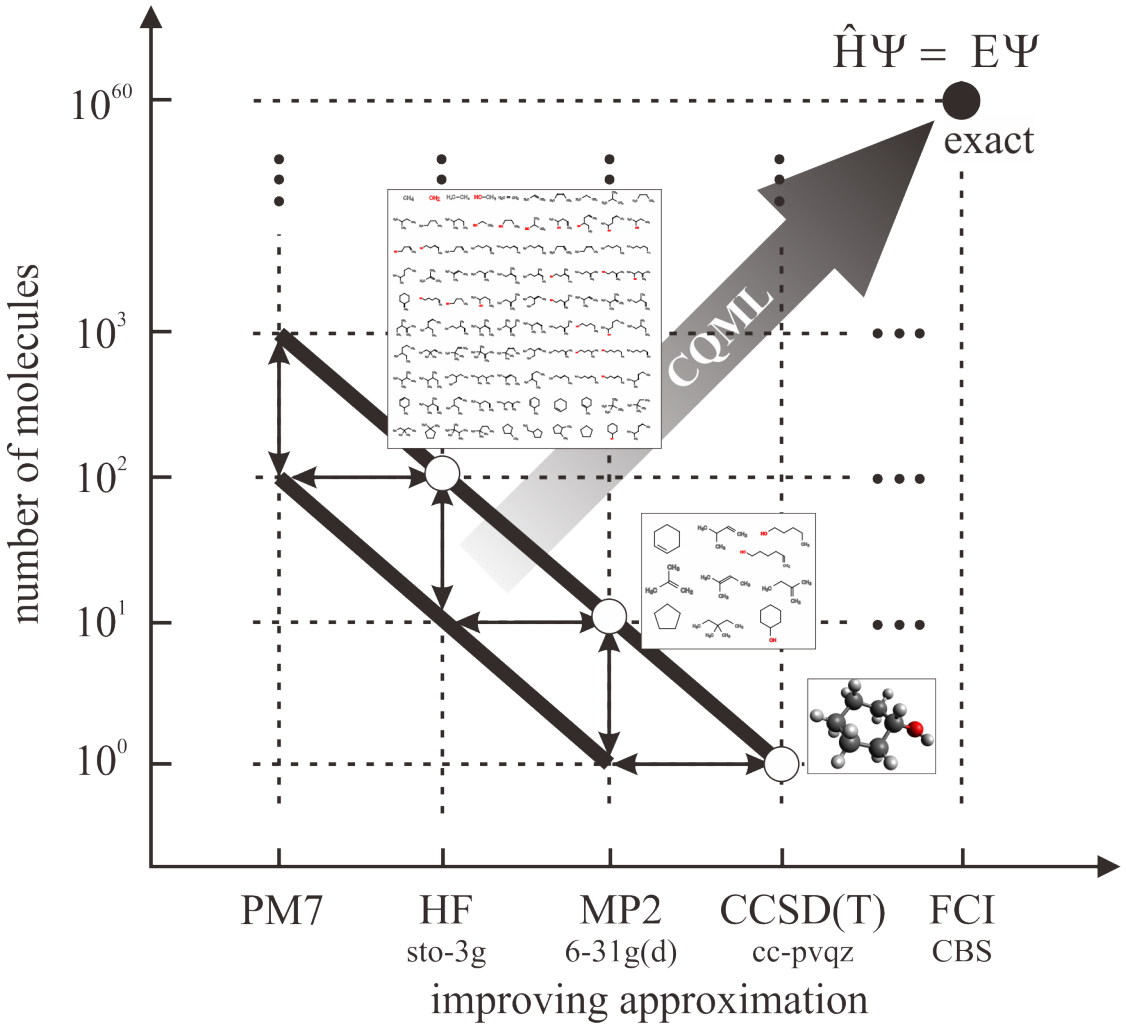


Figure 5.1: Adaptation of Pople diagram involving various levels of theory (abscissa) and molecular spaces (ordinate). The wide arrow indicates how to best approximate highly accurate solutions (solid black circle) of Schrödinger's equation by combining ever improving levels of theory with an exponentially decreasing number of molecules used for training of machine learning models. Qualitative estimates of constant cost-benefit ratios (bold diagonals) correspond to Pareto-optimal solutions which can be sampled using the CQML approach presented herewithin. For example, training data consisting of 1 CCSD(T)/cc-pvqz, 4 MP2/6-31g(d), and 16 HF/sto-3g calculation results can be cheaper *and* more valuable than 3 CCSD(T)/cc-pvqz results. Two-sided arrows indicate bridges in chemical and method space.

correspond to computational protocols which combine various quantum chemical approximations such that high accuracy (frequently chemical accuracy, i.e. $\sim$ 1 kcal/mol) is achieved for thermodynamic quantities (e.g., atomization enthalpies). Typically, they combine the results of a high level of theory with a small basis set with methods that employ lower levels of theory with larger basis sets. Importantly, they impose a computationally much reduced burden when compared to brute-force convergence in basis set size and electron correlations. For example, an extensively used composite method called Gaussian-2 (G2), [CRTP91b] approximates the energy as (starting from a geometry optimized at MP2/6-31g(d) level)

$$E_{\text{true}} \approx E^{\text{G2}} := E_{\text{QCISD(T)/6-311g(d)}} + \Delta_1 + \Delta_2 + \Delta_3, \tag{5.3}$$

where further correction terms have been neglected. Note that here and throughout, we denote approximations and reference results by upper and lower indices, respectively. The individual terms read,

$$\begin{aligned}
\Delta_1 =& E_{\text{MP4/6-311g(2df,p)}} - E_{\text{MP4/6-311g(d)}}, \\
\Delta_2 =& E_{\text{MP4/6-311+g(d,p)}} - E_{\text{MP4/6-311g(d)}}, \\
\Delta_3 =& E_{\text{MP2/6-311+g(3df,2p)}} + E_{\text{MP2/6-311g(d)}} \\
& - E_{\text{MP2/6-311g(2df,p)}} - E_{\text{MP2/6-311g+(d,p)}}
\end{aligned} \tag{5.4}$$

with Δ_1 accounting for the effect of adding the polarization functions, Δ_2 correcting for the diffuse functions and Δ_3 correcting for the larger basis set as well as preventing contributions from being counted twice in Δ_1 and Δ_2, respectively.

Note that the formalism of the composite method corresponds to a sophisticated extension of the telescoping series in Equation (5.2). One could also simply rewrite (5.2) as,

$$E_{\text{CCSD(T)}} = E_{\text{HF}} + \Delta_{\text{HF}}^{\text{MP2}} + \Delta_{\text{MP2}}^{\text{CCSD(T)}}, \tag{5.5}$$

with all terms obtained for some large basis set. The problem then reduces to define efficient yet accurate estimates of the Δs. Here, we introduce the methodology to solve this problem through generalization of the Δ-ML approach [RDRvL15] in the form of CQML.

5.3.2 The CQML approach

To exploit varying levels of theory in order to improve prediction accuracy, and thereby reduce the number of necessary costly training instances some of us previously introduced the Δ-ML approach [RDRvL15]. It uses reference data calculated from a computationally efficient but inaccurate method as a baseline and estimates the difference to a more expensive but accurate target level of theory. Numerical results for organic molecules indicated that given an appropriately chosen baseline method, it is possible to achieve orders of magnitude reduction in training set size when compared to traditional QML approaches. Many other studies have already shown the usefulness and applicability of the Δ-ML approach [FLvLA15, BRvLR17, RHTvL15, RRvL15a, SY18, DOYT17, SR18, SC18, BDP$^+$17b]. Alternatively, efforts have been made towards training set size reduction based on training set optimization [BRvLR17, HvL17a, SNL$^+$18, GPS18] or improvements in the representations [HvL16, HR17, BDP$^+$17a, FCHvL18].

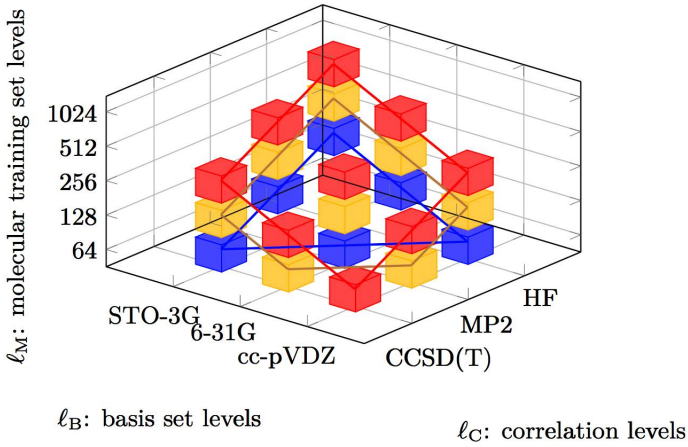


Figure 5.2: The 3D CQML approach combines multiple levels in the spaces of electron correlation, basis sets, and training molecules. Subspaces with $\ell_C + \ell_B + \ell_M = 4$ (see main text for definition of ℓ's) are colored in red, subspaces with $\ell_C + \ell_B + \ell_M = 3$ are colored in yellow and subspaces with $\ell_C + \ell_B + \ell_M = 2$ are colored in blue. They are given in layers, which is indicated by the colored connecting lines.

To the best of our knowledge, no conceptual improvements or generalizations of the Δ-ML approach have been proposed so far.

In this work, we generalize the core ideas of Δ-ML [RDRvL15] to arrive at a multi-level combination technique QML (CQML) approach. CQML is a unified kernel ridge regression machine learning model incorporating training data from several spaces and levels of information. As proposed by e.g. John Pople [Pop65, Pop99], we distinguish between

1. the *space of electron correlation* (e.g. MP2) and
2. the *space of basis set* (e.g. 6-31g), and we also add
3. the *space of training molecules* (e.g. some training set drawn from QM9 [RDRvL14a]) as third degree of freedom which can easily be exploited through machine learning models.

We call a specific choice of training information, e.g Hartree-Fock calculations on a 6-31g basis set done for 256 molecules, a *subspace*. Within each space, we assume a multi-level hierarchy of growing accuracy and computational complexity. E.g. in electron correlation and basis set space, one commonly expects that the degree of approximative nature decays systematically as one goes from HF to MP2 to CCSD(T), or from sto-3g to 6-31g to cc-pvdz, respectively. In chemical space, it is less obvious how to establish a hierarchy of accuracy. For the purpose of our approach, we rely on the well established tenet in statistical learning that the predictive accuracy for out-of-sample molecules increases systematically with training set size [RW06], which is applicable to chemical space and quantum chemistry as demonstrated first in 2012 [RTMvL12a]. This finding has by now been confirmed and reproduced within multiple studies for various quantum properties and system classes [RvL17, vL18]. As such, and when drawing training molecules at random, we can consider their number made available to training (e.g. $N = 16, 32, 64 \ldots$) as the chemical space equivalent to the space of

theory (e.g. HF/MP2/CCSD(T)) or basis set (e.g. sto-3g/6-31g/cc-pvdz). Generally speaking, a CQML model built on low levels of theories / basis sets / small number of training molecules, will result in a model with low accuracy and easily accessible training data. Conversely, including more levels in each dimension, the resulting CQML model will become increasingly more accurate, requiring, however, also access to ever more costly training data. Figure 1 exemplifies these ideas for various levels of electron correlation, basis sets, and molecular training set sizes.

The *sparse grid combination technique* known for high-dimensional approximation[BG04, GSZ92, GH13b, GH13a, GH14, GK00, GK09, HGC07, Pfl97, Rei13, RG18] and quadrature / uncertainty quantification[HPS13b, HPS13a] in numerical analysis corresponds to a rigorous means to generate QML models constructed on a combination of sets of different subspaces. The general idea is to combine the subspaces such that only very few expensive training samples are needed at target accuracy (e.g. CCSD(T) for cc-pvdz at high sample count), some less expensive subspaces with higher training sample count are needed, and so on. Figure 5.2 outlines a choice of subspaces by a modified sparse grid combination technique. Here, each subspace is represented by a colored cube.

In this work, we will first generalize the aforementioned Δ-ML approach to a multi-level approach that incorporates the space of theories, basis sets, and training molecules. The CQML approach differs from existing multi-fidelity machine learning models [PGL17] in that it is (a) generalized to multiple dimensions, and (b) does not unite the various spaces within one kernel matrix, but rather through a series of independently trained kernels. While the CQML approach accounts for an arbitrary number of information spaces, for the sake of brevity and without any loss of generality, we restrict ourselves only to the three spaces discussed above.

5.3.3 Kernel ridge regression and the Δ-ML approach

In order to properly discuss CQML, we first need to briefly recall the principal idea of the established kernel ridge regression based QML models. With $\boldsymbol{R}$ (some) representation of a molecule, we denote by $E_\ell(\boldsymbol{R})$ the ML based approximation of the electronic ground state property of that molecule at a certain level of theory l. We train the ML model using N training molecules $\boldsymbol{R}_i$ with $i = 1, \ldots, N$ with corresponding reference energies at the corresponding specified level, $E_l^{\text{ref}}(\boldsymbol{R})$. The objective is to predict energy E_l^{ref} for an out-of-sample *query* molecule $\boldsymbol{R}_q$, neither part of training nor validation sets.

The ML model E_ℓ within kernel ridge regression is then given by $E_\ell^{\text{ref}}(\boldsymbol{R}_q) \approx E_\ell(\boldsymbol{R}_q) := \sum_{i=1}^{N} \alpha_i^{(\ell)} k(\boldsymbol{R}_q, \boldsymbol{R}_i)$, where k is an appropriate unit-less kernel function. For this study, we always choose the radial basis kernel function, $\exp[-\|\boldsymbol{R}_q - \boldsymbol{R}_i\|_1/\sigma]$ (Laplace) with length-scale σ. Optimization of kernel function space could represent yet another potentially interesting dimension for future investigations. As described in detail elsewhere [RW06, RvL17], the coefficients α_i are obtained by solving the kernel matrix inversion problem $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{e}_\ell^{\text{ref}}$ for given regularizer λ and reference energy vector $\boldsymbol{e}$. Here, we use matrix-notation with capital and small case letters for matrices and vectors, respectively.

The Δ-ML approach [RDRvL15] models the difference between a baseline and target level of theory, e.g. HF and MP2, respectively. Note, that we here have decided to adapt a slightly different notation in contrast to Ref. [RDRvL15] in order to facilitate the generalization of the Δ-ML to the CQML approach. Here, $P_{(b)}(\boldsymbol{R})$ and $P_{(t)}(\boldsymbol{R})$ represent the properties of interest computed at baseline and target level of theory, respectively. Note that within Δ-ML, $P_{(b)}$

and $P_{(t)}$ it is not mandatory to estimate the same property, e.g. it could be the ground state energy in the baseline theory and the enthalpy in the target theory. Hence, the Δ-ML model prediction is given by

$$P_{(t)}(\boldsymbol{R}_q) := P_{(b)}(\boldsymbol{R}_q) + \Delta_b^t(\boldsymbol{R}_q) \tag{5.6}$$

where $\Delta_b^t(\boldsymbol{R}_q) = \sum_{i=1}^{N} \alpha_i k\left(\boldsymbol{R}_q, \boldsymbol{R}_i\right)$. We emphasize that within the Δ-ML model a potentially costly baseline evaluation of the query compound is still necessary when making a prediction. This differs from the CQML approach which recovers the original speed of QML by modeling even the baseline through a machine.

5.3.4 Two-dimensional multi-level learning

The CQML approach generalizes the Δ-ML model to several spaces *and* levels. This is illustrated in Figure 5.2 for three dimensions and levels which we have also considered in this study (*vide infra*). To facilitate the discussion, we first discuss the adaptation of the Pople diagram in order to exemplify the general idea of the CQML approach for the simple case of only two dimensions. More specifically, we now consider the space of theory and training molecules. Thereafter, we will also discuss the generalization to three, as well as d-dimensional cases in Section .

Assuming L levels of theory with running index $\ell = 0, 1, \ldots, L-1$, for which the calculated energy increases in accuracy a (with respect to an experimentally yet unknown truth) and computational cost with growing theoretical complexity, $a^{\ell+1} > a^{\ell}$, for all $\ell < L-1$. Multi-level learning in two dimensions is performed as follows

(1) on level $\ell = 0$ compute reference energies $E^{\text{ref}}_{\ell=0}$ for $N_{\ell=0}$ molecules and train standard QML kernel ridge regression model $E_{\ell=0}$.

(2) on level $\ell = 1$ compute reference energies $E^{\text{ref}}_{\ell=1}$ for $N_{\ell=1} < N_{\ell=0}$ training molecules

(3) Still on level $\ell = 1$, train a model of the difference between E_0 and E^{ref}_1 for the N_1 molecules.

(4) repeat recursively until target level $\ell = L-1$ is reached

Note that while N_ℓ and $N_{\ell+1}$ molecules do not have to be mutual subsets, in this study all $N_{\ell+1}$ molecules are also part of the N_ℓ molecules out of convenience.

Formally, one can recursively define the intermediate multi-level 2D model E_ℓ for $\ell = 0, 1, \ldots, L-1$, and built on the lowest level baseline ($\ell = 0$), as

$$E_\ell(\boldsymbol{R}_q) := E_{\ell-1}(\boldsymbol{R}_q) + \sum_{i}^{N_\ell} \alpha_i^{(\ell-1,\ell)} k\left(\boldsymbol{R}_q, \boldsymbol{R}_i\right) , \tag{5.7}$$

where we set $E^{-1} \equiv 0$. For example, the CQML model which combines PM7 ($\ell = 0$), DFT

($\ell = 1$), and G4MP2 ($\ell = 2$) reads

$$E_2(\boldsymbol{R}_q) = E_1(\boldsymbol{R}_q) + \sum_i^{N_2} \alpha_i^{(1,2)} k(\boldsymbol{R}_q, \boldsymbol{R}_i) \tag{5.8}$$

where

$$E_1(\boldsymbol{R}_q) = E_0(\boldsymbol{R}_q) + \sum_j^{N_1} \alpha_j^{(0,1)} k(\boldsymbol{R}_q, \boldsymbol{R}_j) \tag{5.9}$$

where

$$E_0(\boldsymbol{R}_q) = \sum_k^{N_0} \alpha_k^{(0)} k(\boldsymbol{R}_q, \boldsymbol{R}_k)\,, \tag{5.10}$$

where the last term corresponds to the conventional direct QML model of the PM7 energy. For numerical results obtained from this model, and their discussion *vide infra*. To compute the coefficients $\alpha_i^{(\ell)}$, we solve the previously mentioned kernel ridge regression problem.

Let us briefly compare this approach to the conventional Δ-ML models discussed before in Section 5.3.3. In the single-level case, the resulting model E_0 is the direct conventional QML kernel ridge regression model. In the two-level case, the resulting model E_1 bears similarity with the Δ-ML model, the major difference being that also the baseline is a machine. Thereby, it becomes possible to use different amounts of training information (N_0, N_1) on both levels. Nevertheless, if we chose the training molecules on the first and the second level identical and skipped regularization in the regression problem, E_1 and conventional direct QML would be identical. And if we chose the training molecules on the first and the second level identical and built only one ML model (namely of the difference), E_1^{ref} and Δ-ML would be identical. E_2 and higher order approximations have, to the best of our knowledge, not yet been discussed in the literature.

Using above definition, we did not fix yet how to choose the amount of training samples on each level. This choice is based on the sparse grid combination technique [BG04, GSZ92, GH13b, GH13a, GH14, GK00, GK09, HGC07, Pfl97, Rei13, RG18, HPS13b, HPS13a]. Qualitatively, the combination technique implies to use many training samples on the lower levels of theory and to reduce the number of samples to fewer and fewer samples on higher and higher levels of theory. As we will see, the balance between the amount of training samples per level can be a point of optimization within our method. In Section 5.4, we discuss our choices of level balancing based on the sparse grid combination technique. These choices have been evaluated for different training data, and with respect to two possible optimality measures. Future work will deal with a more systematic assessment of how to tailor and optimize the relative ratios of training molecules at each level and in each dimension.

5.3.5 Three- and d-dimensional multi-level learning

Extending Eq. (5.7) to more than two dimensions results in dimension-dependent levels. Table 5.1 provides an exemplifying overview for three dimensions involving basis set (B), electron correlation (C), and molecular training set (M), with their corresponding levels ℓ_B, ℓ_C and ℓ_M.

Thus, any given combination of levels can be specified as the ordered triplet ℓ of respective level indices, $\ell = (\ell_\text{C}, \ell_\text{B}, \ell_\text{M})$. For example, the combination CCSD(T)/cc-pvdz, N_1 is encoded

Table 5.1: Exemplifying overview of levels in three dimensional multi-level learning for basis sets (B), electron correlation (C), and molecular training set (M).

level	0	1	2
ℓ_C	HF	MP2	CCSD(T)
ℓ_B	sto-3g	6-31g	cc-pvdz
ℓ_M	N_0	N_1	N_2

by the triplet $\ell = (\ell_\mathrm{C} = 2, \ell_\mathrm{B} = 2, \ell_\mathrm{M} = 1) = (2, 2, 1)$.

In order to extend this principle to even more dimensions, we now generalize this approach following the lines of the sparse grid combination technique, cf. Appendix Section 5.6. We introduce for d dimensions corresponding levels $\ell_1, \ldots, \ell_d$, which we collect together in the d-dimensional *multi-index* $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_d)$.

Above, $d = 3$ and ℓ_1 corresponds to ℓ_C, ℓ_2 corresponds to ℓ_B, and ℓ_3 corresponds to ℓ_M. Following the notation that the last level index refers to molecular training set size, i.e. $\ell_d = \ell_\mathrm{M}$, we define the energy $E^\mathrm{ref}_{(\boldsymbol{\ell})}$ given on a subspace $\boldsymbol{\ell}$, and the QML model $E_{\boldsymbol{\ell}}$ for each subspace,

$$E_{\boldsymbol{\ell}}(\boldsymbol{R}_q) := \sum_{i=1}^{N_{\ell_d}} \alpha_i^{(\boldsymbol{\ell})} k\left(\boldsymbol{R}_q, \boldsymbol{R}_i\right) . \tag{5.11}$$

Computing the coefficients $\alpha_i^{(\boldsymbol{\ell})}$ for a fixed subspace $\boldsymbol{\ell}$ is done by solving the regression problem

$$E^\mathrm{ref}_{(\boldsymbol{\ell})}(\boldsymbol{R}_j) \approx \sum_{i=1}^{N_{\ell_d}} \alpha_i^{(\boldsymbol{\ell})} k\left(\boldsymbol{R}_j, \boldsymbol{R}_i\right) \tag{5.12}$$

for all $j = 1, \ldots, N_{\ell_d}$.

The generalized CQML machine learning model is then given as

$$E_{\mathcal{I}}(\boldsymbol{R}_q) := \sum_{\boldsymbol{\ell}\in\mathcal{I}} \beta_{\boldsymbol{\ell}} \sum_{i=1}^{N_{\ell_d}} \alpha_i^{(\boldsymbol{\ell})} k\left(\boldsymbol{R}_q, \boldsymbol{R}_i\right) . \tag{5.13}$$

In fact, it is the combination of the machine learning models from eq. 5.11 for different subspaces $\boldsymbol{\ell}$ that are collected in the *index set* $\mathcal{I}$. The classical sparse grid combination technique proposes to use the index set

$$\mathcal{I} := \{\boldsymbol{\ell} \in \mathbb{N}^d | \|\boldsymbol{\ell}\|_1 = (L-1) - i, i \in \{0, \ldots, d-1\}\} \tag{5.14}$$

with $\|\boldsymbol{\ell}\|_1 := \sum_{s=1}^d \ell_s$. In the following, the coefficients $\beta_{\boldsymbol{\ell}}$ can always be evaluated as[RG18]

$$\beta_{\boldsymbol{\ell}} := \sum_{\boldsymbol{z}\in\{0,1\}^d} (-1)^{\|\boldsymbol{z}\|_1} \chi_{\mathcal{I}}(\boldsymbol{\ell} + \boldsymbol{z}) . \tag{5.15}$$

Here, the sum is to be understood in the sense that vector $\boldsymbol{z}$ of size d takes all possible combinations of zeros and ones. Moreover we define the characteristic function $\chi_{\mathcal{I}}$ of index set

$\mathcal{I}$ by

$$\chi_{\mathcal{I}}(\boldsymbol{\ell}+\boldsymbol{z}) := \begin{cases} 1 & \text{if } (\boldsymbol{\ell}+\boldsymbol{z}) \in \mathcal{I}, \\ 0 & \text{else.} \end{cases} \tag{5.16}$$

It is well-known [] that the above choice of the index set $\mathcal{I}$ and coefficients $\beta_{\boldsymbol{\ell}}$ in $d = 2$ is equivalent to the multi-level learning approach from Section 5.3.4.

For $d = 3$ the above choice of index set $\mathcal{I}$ would lead to the CQML model $E_{(2,2,2)}$ with

$$\begin{aligned} E_{(2,2,2)}(\boldsymbol{R}_q) =& E_{(0,2,0)}(\boldsymbol{R}_q) - 2E_{(0,1,0)}(\boldsymbol{R}_q) + E_{(1,1,0)}(\boldsymbol{R}_q) + E_{(0,1,1)}(\boldsymbol{R}_q) - 2E_{(1,0,0)}(\boldsymbol{R}_q) \\ &+ E_{(0,0,0)}(\boldsymbol{R}_q) - 2E_{(0,0,1)}(\boldsymbol{R}_q) + E_{(2,0,0)}(\boldsymbol{R}_q) + E_{(1,0,1)}(\boldsymbol{R}_q) + E_{(0,0,2)}(\boldsymbol{R}_q) \end{aligned} \tag{5.17}$$

and $\ell_{\mathrm{C}}, \ell_{\mathrm{B}}, \ell_{\mathrm{M}} = 0, \ldots, 2$, exemplified with the spaces discussed in Section 5.1. Note, however, that this choice does not use any training data from the target subspace, here CCSD(T) calculations with a cc-pvdz basis set. In practice, it is preferable to include the corresponding subspaces with this accuracy to the training set, at least with a small training set size, in order to include the physics of the corresponding target subspace. To this end, in $d = 3$, we shift the index set $\mathcal{I}$ such that the subspace choice from Figure 5.2 is achieved. The resulting index set becomes

$$\mathcal{I}_{shifted} := \{(\ell_C, \ell_B, \ell_M) | \ell_C, \ell_B \in [0,2], \ell_M \in [0,4], \ell_C + \ell_B + \ell_M \in \{2,3,4\}\} \tag{5.18}$$

The corresponding CQML model then becomes $E^{shifted}_{(2,2,2)}$, cf. Appendix Section 5.6 and reads

$$\begin{aligned} E^{shifted}_{(2,2,2)}(\boldsymbol{R}_q) =& -2E_{(1,2,0)}(\boldsymbol{R}_q) + E_{(0,2,0)}(\boldsymbol{R}_q) - 2E_{(0,2,1)}(\boldsymbol{R}_q) + E_{(2,2,0)}(\boldsymbol{R}_q) \\ &+ E_{(1,2,1)}(\boldsymbol{R}_q) + E_{(0,2,2)}(\boldsymbol{R}_q) - 2E_{(2,1,0)}(\boldsymbol{R}_q) + E_{(1,1,0)}(\boldsymbol{R}_q) \\ &- 2E_{(1,1,1)}(\boldsymbol{R}_q) + E_{(0,1,1)}(\boldsymbol{R}_q) - 2E_{(0,1,2)}(\boldsymbol{R}_q) + E_{(2,1,1)}(\boldsymbol{R}_q) \\ &+ E_{(1,1,2)}(\boldsymbol{R}_q) + E_{(0,1,3)}(\boldsymbol{R}_q) + E_{(2,0,0)}(\boldsymbol{R}_q) - 2E_{(2,0,1)}(\boldsymbol{R}_q) \\ &+ E_{(1,0,1)}(\boldsymbol{R}_q) - 2E_{(1,0,2)}(\boldsymbol{R}_q) + E_{(0,0,2)}(\boldsymbol{R}_q) - 2E_{(0,0,3)}(\boldsymbol{R}_q) \\ &+ E_{(2,0,2)}(\boldsymbol{R}_q) + E_{(1,0,3)}(\boldsymbol{R}_q) + E_{(0,0,4)}(\boldsymbol{R}_q) \end{aligned} \tag{5.19}$$

with $\ell_{\mathrm{C}}, \ell_{\mathrm{B}}, \ell_{\mathrm{M}} = 0, \ldots, 2$. The reader is referred to Appendix Section 5.6 for more details of the CQML derivation.

5.4 Results and discussion

Before entering the detailed discussion of our results, we now briefly discuss the use of learning curves as a measure of machine learning model quality. Clearly, reporting a single out-of-sample error for any machine learning model is hardly meaningful: It is the very point of machine learning that models should improve with training set size. Vapnik and co-workers discussed already in the nineties that prediction errors, i.e. out-of-sample estimates of statistically estimated functions, decay inversely with training set size N. More specifically, for kernel

ridge regression models (used throughout this study), the leading prediction error term was shown to be proportional to a/N^b, where a and b are proportionality constant and power law exponent, respectively [CJS+94, MFM+96, Vap13]. In order to facilitate comparison among models, it is therefore recommended practice [vL18] to discuss the performance in terms of learning curves on log-log scales, i.e. for prediction errors decaying linearly with training set size, i.e. $\log(\text{Error}) = \log(a) - b\log(N)$. Saturation of errors indicates failure to learn; and small off-sets and steep slopes indicate preferable models.

5.4.1 Data

For all the ∼7'000 QM7b molecules [MRG+13a], we have calculated total energies for all combinations among the various levels of correlation energies (HF, MP2, CCSD(T)) and basis set sizes (sto-3g, 6-31g, cc-pvdz). Resulting effective atomization energies (see SI for the entire data set), are shown within scatter-plots in Fig. 5.3. Depending on stoichiometry and size, the molecules spread out over the various levels and dimensions.

More specifically, molecules can be divided into two clusters: the one dominating the distribution is almost sulfur-free; while the other cluster of molecules, clearly separated from the majority, contains sulfur atoms (see bottom row in Fig. 5.3). This pattern indicates that sto-3g and 6-31g are too small basis sets, and should not be used to describe S containing molecules. By comparing the three figures in each column of the first three rows in Figure 5.3, one can see that the shape of distribution changes significantly upon introduction of electron correlation (going from HF treatment to the MP2). When going from MP2 to CCSD(T), however, the change in the distribution is barely noticeable.

Considering the right hand panel in the third row in Fig. 5.3, the color code corresponds exactly to the correlation energy contribution to the atomization energy, as estimated by CCSD(T) - HF within cc-pvdz basis. As one would expect, the larger the molecule, the more electron correlation energy is being contributed. The two hundred molecules with the largest and smallest correlation energy contribution to the atomization energy are on display in Fig. 5.4. We note that molecules with high degree of saturation exhibit the largest amount of electron correlation in their atomization energy, while atomization energies of molecules with multiple double bonds, triple bonds, and aromatic moieties contain the least electron correlation energy. This trend is to be expected because the electrons in unsaturated bonding patterns can contribute less to binding than in saturated species, thereby also decreasing their electron correlation energy contribution to binding.

The reason for developing the CQML model is based on the hypothesis that it will systematically exploit all these underlying implicit correlations which are on display in these figures.

5.4.2 2D results for QM7b

As a first test, we have investigated our QM7b derived data set for the two dimensional ($d = 2$) case of atomization energies at a fixed basis set (cc-pvdz) for three levels of electron correlation, i.e. HF ($\ell_C = 0$), MP2 ($\ell_C = 1$) and CCSD(T) ($\ell_C = 2$). The second dimension corresponds to three variable molecular training data set sizes ($\ell_M = 0, 1, 2$). Their relative extent is fixed at ratios which are independent of absolute training set size. In this study, we considered two such sets of ratios ($s = 1$ and $s = 2$) which reflect different sample size increases for higher levels.

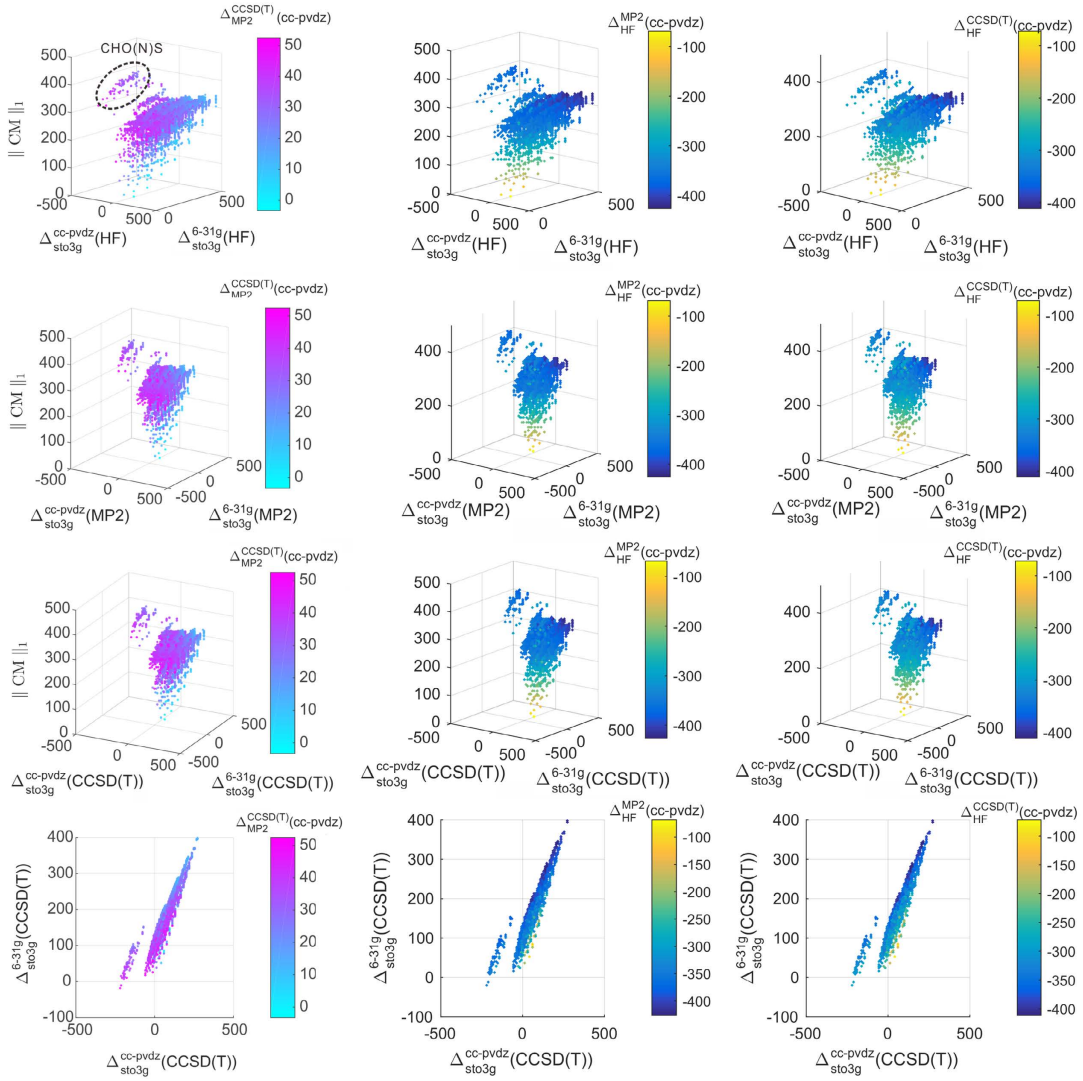


Figure 5.3: Scatter plots for QM7b. Size in chemical space as measured by 1-norm of Coulomb matrix [a.u.] (i.e., $\|\mathrm{CM}\|_1$) vs. energy differences [kcal/mol] due to various basis set size differences for HF (first row), MP2 (second row), and CCSD(T) (third row). The colour code corresponds to the atomization energy difference Δ [kcal/mol] between electron correlation models at cc-pvdz for MP2 vs. CCSD(T) (left), HF vs. MP2 (mid), and HF vs. CCSD(T). In the upper leftmost panel, the brackets enclosing N indicate that nitrogen atoms may or may not be present. The bottom row corresponds to the 2D projection of the third row.

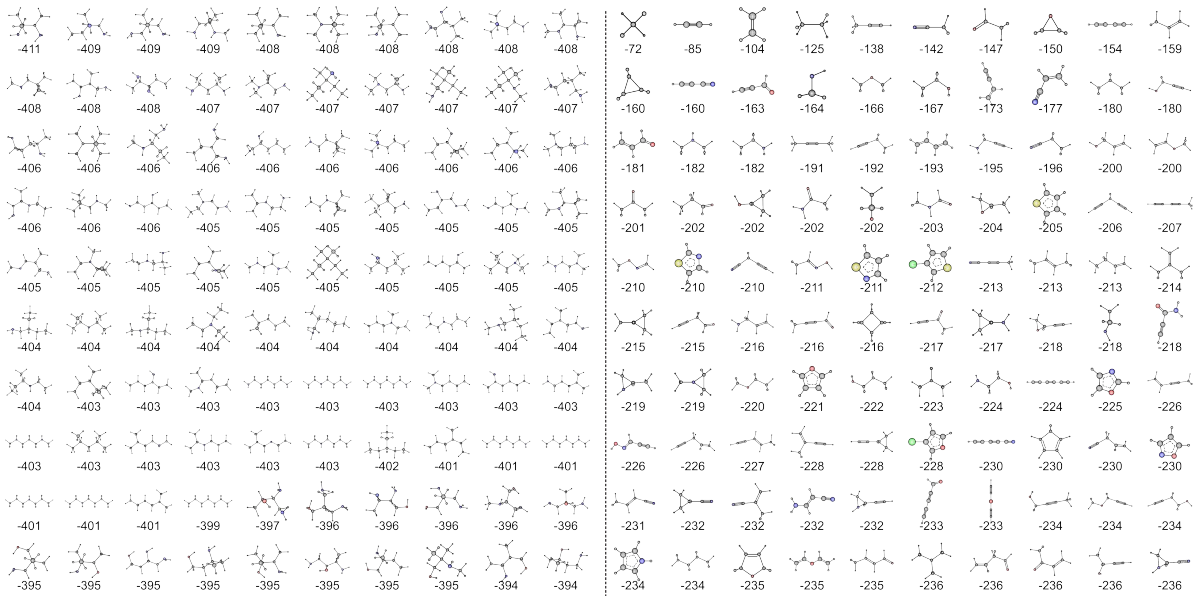


Figure 5.4: The two hundred QM7b molecules with largest (Left) and smallest (Right) electron correlation energy contributions to the atomization energy (CCSD(T) - HF within cc-pvdz basis [kcal/mol]), respectively. See SI for the complete data set. White: H, gray: C, yellow: S, red: O, blue: N, green: Cl.

Table 5.2: Level-dependent ratios between training set sizes for the two sample size increases s considered. L is the total number of levels.

s	$r_{\ell_\mathrm{M}=L-1}$	$r_{\ell_\mathrm{M}=L-2}$	$r_{\ell_\mathrm{M}=L-3}$
1	1	2	4
2	1	4	16

These ratios are summarized in Table 5.2. The number of training molecules N_{ℓ_M} on each level of the CQML with $d = 2$ as a function of training set size at the highest level $N_{\ell_\mathrm{M}=2}$ is thus given by $N_{\ell_\mathrm{M}} = r_{\ell_\mathrm{M}} \times N_{\ell_\mathrm{M}=2}$, where r_{ℓ_M} is the ratio as displayed in Table 5.2. Recall that all ML model results presented in this section have been obtained using kernel ridge regression, a Laplacian kernel, and the SLATM [HvL17a] representation.

In Figure 5.5, various learning curves for atomization energies, estimated according to Eq. 5.7, are shown. First of all, we note the rapid and systematic lowering for all CQML models as training set size increases. The models exhibit differing off-sets, and similar slopes, in line with previous results for training-set optimization experiments using ensembles of training sets within genetic optimization protocols [BRvLR17]. The learning curves of conventional QML pass the chemical accuracy threshold ($\sim$1 kcal/mol) at $\sim 4'000$ training molecules calculated at target level, CCSD(T)/cc-pvdz. This learning curve has a slightly larger off-set with respect to the original SLATM benchmark results (see supplementary materials in Ref. [HvL17b]) due to the use of (i) the Laplacian instead of a Gaussian kernel function, (ii) B3LYP rather than PBE0 geometries, and (iii) CCSD(T) rather than PBE0 energies.

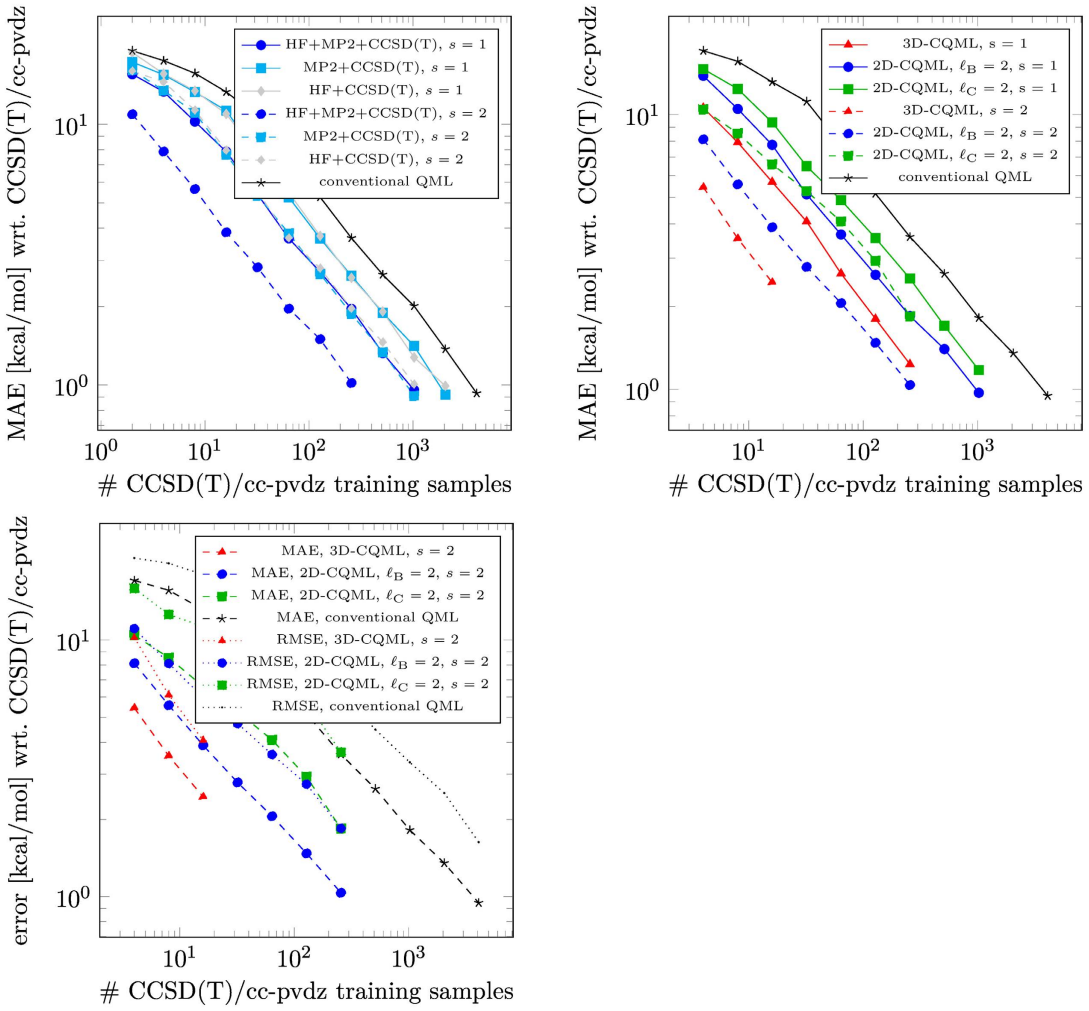


Figure 5.5: Learning curves for CCSD(T) atomization energies of QM7b molecules for various CQML models. Level ratios considered include $s = 1$ and $s = 2$ (See Table 5.2). Upper left: 2D-CQML at fixed basis set (cc-pvdz) including 2 (MP2, CCSD(T))/(HF,CCSD(T)), and 3 levels of electron correlation treatment (HF, MP2, CCSD(T)). Upper right: 2D-CQML (green) at fixed electron correlation treatment (CCSD(T)) for 3 basis sets (sto-3g, 6-31g, cc-pvdz). 3D-CQML (red) exploiting basis set size (sto-3g, 6-31g, cc-pvdz) *and* electron correlation treatment (HF, MP2, CCSD(T)). Bottom: Learning curves for RMSE (root mean square error) and MAE for the machines in the Upper right panel with $s = 2$. Note that the horizontal axis in all three figures may also be chosen to represent the number of training samples from other levels, which can be obtained by rescaling the current axis with a ratio of s, as shown in Table 5.2.

Addition of MP2 reference energies of further molecules affords a systematic decrease in the learning off-set resulting in ∼2'000 and ∼1'000 CCSD(T) training molecules necessary to reach chemical accuracy for $s = 1$ and $s = 2$, respectively. The corresponding necessary MP2 training set sizes (not shown in the figure) amount to 4'000 molecules for both s-values (see Table 5.2). Slightly worse results are obtained by replacing MP2 reference energies with HF energies. This result may seem puzzling, but is in full agreement with what we have found in Figure 5.3, i.e., the values of $\Delta_{\mathrm{MP2}}^{\mathrm{CCSD(T)}}$ and $\Delta_{\mathrm{HF}}^{\mathrm{MP2}}$ are of the same magnitude. This result also implies the possibility to optimize the levels of theory by minimizing the computational cost, meanwhile retaining the accuracy.

Adding Hartree-Fock treatment for additional training molecules, we observe even further improvement, reaching chemical accuracy already at ∼1'000 and ∼300 CCSD(T) training molecules for $s = 1$ and $s = 2$, respectively. According to the ratios in Table 5.2, the corresponding necessary MP2 and HF training set sizes (not shown in the figure) amount respectively to 2'000 and 4'000 for $s = 1$, and to 1'200 and 2'400 for $s = 2$.

These results are very encouraging; they suggest that reductions by an order of magnitude are possible with respect to high-level reference numbers (from expensive computation or experiment) necessary to reach chemical accuracy. Effectively, the CQML model appears to exploit correlations inherent among the various approximation levels that live within hierarchical spaces of theories.

5.4.3 3D results for QM7b

We have also studied the extension of the 2D-CQML model by a third dimension ($d = 3$) which explicitly introduces the effect of basis set size on atomization energies. More specifically, we have considered sto-3g ($\ell_{\mathrm{B}} = 0$) as our lowest level, 6-31g ($\ell_{\mathrm{B}} = 1$) as an intermediate size, and cc-pvdz ($\ell_{\mathrm{B}} = 2$) as the largest set. Obviously, larger basis set choices as well as additional levels with more subtle differences could have been included just as well. Here, we assume that the general trend and the conclusions drawn are not affect by the relatively modest size of the basis sets employed.

In Fig. 5.5, we show corresponding learning curves of 2D-CQML models which connect the different basis sets according to Eq. 5.7 with just one correlation energy model, CCSD(T). In line with the behavior encountered above for the fixed basis set CQML models, a systematic improvement is found for MAE as well as RMSE. The error approaches chemical accuracy already with ∼1'000 training examples with the largest basis used (cc-pvdz). Again, increasing the ratios between levels by going from $s = 1$ to $s = 2$ (see Table 5.2) leads to systematic lowering of the learning curve.

Finally, when combining multiple basis set and electron correlation levels into a single 3D-CQML model, obtained according to Eq. 5.19, the most favorable learning curves are obtained for MAE as well as for RMSE (See Fig. 5.5). For $s = 1$ and $s = 2$, extrapolation indicates that chemical accuracy can be reached with just 500 and 100 training instances at CCSD(T)/cc-pvdz level, respectively. Note that the learning curves end already for relatively small training set sizes because the necessary number of molecules required at lower levels of theory rapidly reaches the maximal number of available molecules in QM7b. For example, for the $s = 2$ case, 100 training molecules at the highest level combination would have required $100 \times 4^4 = 25{,}600$ training molecules at the lowest level combination. However, QM7b is comprised of only 7'211

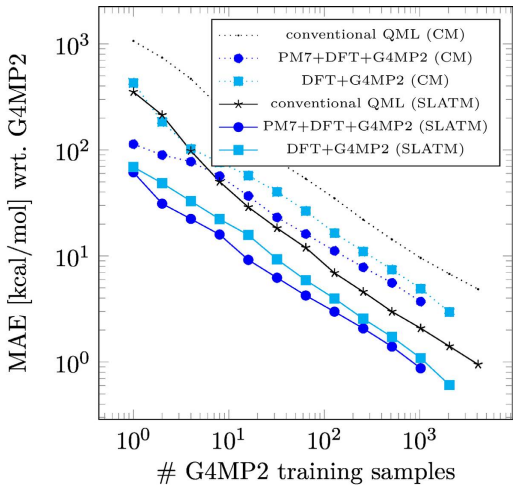


Figure 5.6: Prediction errors of atomization energies in CI9 data set (consitutional isomers of $C_7H_{10}O_2$) vs. number of training molecules with G4MP2 energies for various 2D-CQML models. Results differ by representation (SLATM vs. CM) and number of levels included.

molecules. As such, this is an artefact of the finite size of QM7b, and we expect these learning curves to further decay linearly when using larger data sets in the future.

Overall, these results amount to numerical evidence that it is beneficial to include not only multiple levels but also multiple dimensions. The obvious consequence is that an additional substantial reduction in need for high-level reference numbers (from expensive computation or experiment) is possible through the use of CQML based exploitation of training data obtained for smaller basis sets and more approximate electron correlation models. We believe that this is possible because of inherent error-cancellation between various levels *and* dimensions.

Furthermore, the success of CQML indicates that we can push the efforts further so as to achieve an absolute prediction error on par with experiments by taking advantage of even higher level theory of reference data, such as CCSD(T)-F12/cc-pvqz or Quantum Monte Carlo (QMC). More specifically, the highest level of theory adopted in this study is CCSD(T)/cc-pvdz, to which the prediction errors are referenced. As a result, the predictive power of CQML can match at best CCSD(T)/cc-pvdz. Considering that we need very few training instances at the highest level, we are optimistic that at a very low cost (cf. brute force high-level-of-theory calculation), we can eventually achieve chemical accuracy for energy predictions of arbitrary out-of-sample molecules. This will be explored in future work.

5.4.4 2D results for CI9

For the stoichiometrical isomers $C_7H_{10}O_2$, data set CI9, we have also investigated the 2D-CQML model corresponding to Eq. 5.7. The resulting models differ from the previous 2D-CQML models in that they unite energy approximation effects and basis sets into a single dimension (PM7, B3LYP/6-31g(d), G4MP2). Furthermore, and in analogy to the original Δ-ML model [RDRvL15, SNZ$^+$18, RRvL15b, RHTvL15, BDJTVL18, KO00], all small changes

in geometry due to use of different level of theory, are also being accounted for through the ML model. As such, only PM7-quality input geometries are required for the 2D-CQML models discussed in this section. Resulting learning curves are shown in Fig. 5.6 for two different representations, the Coulomb matrix [RTMvL12a, HMB$^+$13] and SLATM [HvL17b], as well as for two different number of levels ($L = 2$ and $L = 3$).

Again, when compared to conventional QML, we note systematic and improved (through lower off-sets) learning as the number of different levels increases from two to three. The relative performance for Coulomb matrix and SLATM meets the expected trend [FCHvL18], SLATM systematically leading to a substantially lower off-set. These results suggest a certain independence of the CQML methodology from other salient features of QML models, such as training set selection [BRvLR17, HvL17b] or choice of representation [HvL16, FCHvL18]. In this case, the best 2D-CQML SLATM based model reaches chemical accuracy with respect to G4MP2 based on a training set consisting of ∼1'000, 2'000, and 4'000 at G4MP2, B3LYP/6-31g(d), and PM7 level reference results, respectively.

5.5 Conclusions

We have extended the ideas manifested in Pople-diagrams within the systematic framework of the multi-level sparse grid combination technique and machine learning. A generalized CQML model has been presented, and we have demonstrated its performance for various 2D variants and for one 3D application using atomization energies of organic molecules as property of interest. Using learning curves to compare models, we have found for all cases investigated that the addition of levels and spaces enables a systematic and substantial reduction in necessary training data at the highest level of theory. As such, we have shown how to construct QML models for which an expensive training molecule can be replaced by multiple cheaper training molecules. Due to the unfavourable polynomial scaling and large prefactors of the more expensive quantum approximations, such trade-offs can deliver significantly more accurate QML models at constant training data compute budget. In conclusion, our numerical findings support the idea that there is an additional "knob" one can use to improve QML models: In addition to improved representations [HvL16, FCHvL18] or training set selection [BRvLR17, HvL17b] one can also exploit the intrinsic correlations among the various hierarchies which exist among different levels of approximations.

For future work, we will consider the inclusion of more intermediate levels, e.g. the various rungs on Jacob's ladder, or MP4, CCSD, CCSDT(Q), etc., or continuous changes in basis set size through plane-waves. Other dimensions, such as relativistic effects, spin-orbit coupling, or nuclear quantum effects can be envisioned. While we have focussed on atomization energies only for this study, we will consider CQML models of other quantum properties within subsequent studies. Technical settings can also be investigated, e.g. the relative amount of training data obtained at different levels (currently set globally through parameter s), could still be adapted in a locally optimal manner. Finally, we plan to include this implementation in qmlcode [CFH$^+$17].

5.6 Appendix: Derivation of the combination technique for quantum machine learning

In applied mathematics, the sparse grid combination technique is a means to approximate, e.g., high-dimensional functions. Lets assume that such a function f is in some (function) space $V := V^{(1)} \otimes V^{(2)} \otimes \cdots \otimes V^{(d)}$. That is, it is in the tensor product of d spaces. Then, we introduce for each of the L_m-dimensional function spaces $V^{(m)}$ a series of subspaces of lower dimension

$$V_0^{(m)} \subset V_1^{(m)} \subset V_j^{(m)} \subset \ldots \subset V_{L_m}^{(m)} \tag{5.20}$$

(indicated by the lower index). Classic (full tensor product) approximation would now approximate this function f on a level j in the space $V_j := V_j^{(1)} \otimes V_j^{(2)} \otimes \cdots \otimes V_j^{(d)}$. However, this leads to the so-called *curse of dimensionality*, i.e. the exponential growth in computational work with growing dimension d.

In many cases, the sparse grid combination technique allows to approximate f in a much cheaper way. This is done by recursively introducing the sparse approximation space $\hat{V}_j$ with

$$\hat{V}_j^{(d)} := \sum_{k=0}^{j} \left(V_{j-k}^{(d)} - V_{j-1-k}^{(d)} \right) \otimes \hat{V}_k^{(d-1)} , \tag{5.21}$$

where $\hat{V}_k^{(d-1)}$ is the sparse approximation space

$$\hat{V}_j^{(d-1)} := \sum_{k=0}^{j} \left(V_{j-k}^{(d-1)} - V_{j-1-k}^{(d-1)} \right) \otimes \hat{V}_k^{(d-2)} . \tag{5.22}$$

That is, it is recursively built from the first $d-1$ spaces in the same way.

In this work, we transfer this approach to the field of quantum machine learning. To this end, we provide a derivation for the combination technique for quantum machine learning in two and three dimensions / spaces. Let us first briefly introduce a general machine learning model for a given subspace (ℓ_C, ℓ_B, ℓ_M). Note that we assume here that $\ell_C, \ell_B, \ell_M \in \{0, \ldots, L\}$. The general ML model for a given subspace reads as

$$E_{(\ell_C,\ell_B,\ell_M)}(\boldsymbol{R}_q) := \sum_{i=1}^{N_{\ell_M}} \alpha_i^{(\ell_C,\ell_B,\ell_M)} k(\boldsymbol{R}_q, \boldsymbol{R}_i) . \tag{5.23}$$

We identify this model with some subspace $V_{\ell_C}^{(1)} \otimes V_{\ell_B}^{(2)} \otimes V_{\ell_M}^{(3)}$. Following equation 5.21, the two-dimensional combination technique for QML on level j_2 for the spaces of theory and training set size and a fixed basis set level ℓ_B can be introduced as

$$E_{(j_2,\ell_B,j_2)}(\boldsymbol{R}_q) := \sum_{k_2=0}^{j_2} E_{(j_2-k_2,\ell_B,k_2)}(\boldsymbol{R}_q) - E_{(j_2-1-k_2,\ell_B,k_2)}(\boldsymbol{R}_q) \tag{5.24}$$

Note that, whenever a level index becomes negative, we assume the machine learning model to

be exactly zero, i.e. $E_{(-1,\cdot,\cdot)} \equiv E_{(\cdot,-1,\cdot)} \equiv E_{(\cdot,\cdot,-1)} \equiv 0$. For the choice of $j_2 = 2$ and $\ell_B = 2$, we can explicitly derive

$$\begin{aligned} E_{(2,2,2)}(\boldsymbol{R}_q) &= \big(E_{(2-0,2,0)}(\boldsymbol{R}_q) - E_{(2-1-0,2,0)}(\boldsymbol{R}_q)\big) + \big(E_{(2-1,2,1)}(\boldsymbol{R}_q) - E_{(2-1-1,2,1)}(\boldsymbol{R}_q)\big) \\ &\quad + \big(E_{(2-2,2,2)}(\boldsymbol{R}_q) - E_{(2-1-2,2,2)}(\boldsymbol{R}_q)\big) \\ &= \big(E_{(2,2,0)}(\boldsymbol{R}_q) - E_{(1,2,0)}(\boldsymbol{R}_q)\big) + \big(E_{(1,2,1)}(\boldsymbol{R}_q) - E_{(0,2,1)}(\boldsymbol{R}_q)\big) \\ &\quad + \big(E_{(0,2,2)}(\boldsymbol{R}_q) - E_{(-1,2,2)}(\boldsymbol{R}_q)\big) \\ &= E_{(2,2,0)}(\boldsymbol{R}_q) - E_{(1,2,0)}(\boldsymbol{R}_q) + E_{(1,2,1)}(\boldsymbol{R}_q) - E_{(0,2,1)}(\boldsymbol{R}_q) + E_{(0,2,2)}(\boldsymbol{R}_q) \end{aligned} \tag{5.25}$$

Note that we have the equalities

$$\begin{aligned} E_{(2,2,0)}(\boldsymbol{R}_q) - E_{(1,2,0)}(\boldsymbol{R}_q) &= \sum_i^{N_2} \alpha_i^{(1,2)} k(\boldsymbol{R}_q, \boldsymbol{R}_i)\,, \\ E_{(1,2,1)}(\boldsymbol{R}_q) - E_{(0,2,1)}(\boldsymbol{R}_q) &= \sum_i^{N_1} \alpha_i^{(0,1)} k(\boldsymbol{R}_q, \boldsymbol{R}_i)\,, \\ E_{(0,2,2)}(\boldsymbol{R}_q) &= \sum_i^{N_0} \alpha_i^{(0)} k(\boldsymbol{R}_q, \boldsymbol{R}_i)\,, \end{aligned} \tag{5.26}$$

with the notation from Section 5.3.4. That is, model $E_{(2,2,2)}$, as derived here, is exactly the model discussed in Section 5.3.4.

Based on the two-dimensional combination technique model, we can now recursively build a three-dimensional combination technique further integrating the space of basis set size and with the global three-dimensional level j_3 as follows

$$E_{(j_3,j_3,j_3)}(\boldsymbol{R}_q) := \sum_{k_3=0}^{j_3} E_{(k_3,j_3-k_3,k_3)}(\boldsymbol{R}_q) - E_{(k_3,j_3-1-k_3,k_3)}(\boldsymbol{R}_q)\,. \tag{5.27}$$

This construction uses the definition of the two-dimensional combination technique in a recursive fashion.

We finally exemplify the tree-dimensional combination technique for $j_3 = 2$. That is, we first expand the recursive model for the three-dimensional combination technique by

$$\begin{aligned} E_{(2,2,2)}(\boldsymbol{R}_q) &= \big(E_{(0,2-0,0)}(\boldsymbol{R}_q) - E_{(0,2-1-0,0)}(\boldsymbol{R}_q)\big) + \big(E_{(1,2-1,1)}(\boldsymbol{R}_q) - E_{(1,2-1-1,1)}(\boldsymbol{R}_q)\big) \\ &\quad + \big(E_{(2,2-2,2)}(\boldsymbol{R}_q) - E_{(2,2-1-2,2)}(\boldsymbol{R}_q)\big) \\ &= \big(E_{(0,2,0)}(\boldsymbol{R}_q) - E_{(0,1,0)}(\boldsymbol{R}_q)\big) + \big(E_{(1,1,1)}(\boldsymbol{R}_q) - E_{(1,0,1)}(\boldsymbol{R}_q)\big) \\ &\quad + \big(E_{(2,0,2)}(\boldsymbol{R}_q) - E_{(2,-1,2)}(\boldsymbol{R}_q)\big) \\ &= \big(E_{(0,2,0)}(\boldsymbol{R}_q) - E_{(0,1,0)}(\boldsymbol{R}_q)\big) + \big(E_{(1,1,1)}(\boldsymbol{R}_q) - E_{(1,0,1)}(\boldsymbol{R}_q)\big) + E_{(2,0,2)}(\boldsymbol{R}_q)\,. \end{aligned} \tag{5.28}$$

Then, we expand each of the term by means of the two-dimensional combination technique.

Thus we compute

$$
\begin{aligned}
E_{(0,2,0)}(\boldsymbol{R}_q) &= E_{(0-0,2,0)}(\boldsymbol{R}_q) - E_{(0-1-0,2,0)}(\boldsymbol{R}_q) = E_{(0,2,0)}(\boldsymbol{R}_q)\,, \\
E_{(0,1,0)}(\boldsymbol{R}_q) &= E_{(0-0,1,0)}(\boldsymbol{R}_q) - E_{(0-1-0,1,0)}(\boldsymbol{R}_q) = E_{(0,1,0)}(\boldsymbol{R}_q)\,, \\
E_{(1,1,1)}(\boldsymbol{R}_q) &= E_{(1-0,1,0)}(\boldsymbol{R}_q) - E_{(1-1-0,1,0)}(\boldsymbol{R}_q) + E_{(1-1,1,1)}(\boldsymbol{R}_q) - E_{(1-1-1,1,1)}(\boldsymbol{R}_q) \\
&= E_{(1,1,0)}(\boldsymbol{R}_q) - E_{(0,1,0)}(\boldsymbol{R}_q) + E_{(0,1,1)}(\boldsymbol{R}_q)\,, \\
E_{(1,0,1)}(\boldsymbol{R}_q) &= E_{(1-0,0,0)}(\boldsymbol{R}_q) - E_{(1-1-0,0,0)}(\boldsymbol{R}_q) + E_{(1-1,0,1)}(\boldsymbol{R}_q) - E_{(1-1-1,0,1)}(\boldsymbol{R}_q) \\
&= E_{(1,0,0)}(\boldsymbol{R}_q) - E_{(0,0,0)}(\boldsymbol{R}_q) + E_{(0,0,1)}(\boldsymbol{R}_q)\,, \\
E_{(2,0,2)}(\boldsymbol{R}_q) &= \big(E_{(2-0,0,0)}(\boldsymbol{R}_q) - E_{(2-1-0,0,0)}(\boldsymbol{R}_q)\big) + \big(E_{(2-1,0,1)}(\boldsymbol{R}_q) - E_{(2-1-1,0,1)}(\boldsymbol{R}_q)\big) \\
&\quad + \big(E_{(2-2,0,2)}(\boldsymbol{R}_q) - E_{(2-1-2,0,2)}(\boldsymbol{R}_q)\big) \\
&= \big(E_{(2,0,0)}(\boldsymbol{R}_q) - E_{(1,0,0)}(\boldsymbol{R}_q)\big) + \big(E_{(1,0,1)}(\boldsymbol{R}_q) - E_{(0,0,1)}(\boldsymbol{R}_q)\big) \\
&\quad + \big(E_{(0,0,2)}(\boldsymbol{R}_q) - E_{(-1,0,2)}(\boldsymbol{R}_q)\big) \\
&= \big(E_{(2,0,0)}(\boldsymbol{R}_q) - E_{(1,0,0)}(\boldsymbol{R}_q)\big) + \big(E_{(1,0,1)}(\boldsymbol{R}_q) - E_{(0,0,1)}(\boldsymbol{R}_q)\big) + E_{(0,0,2)}(\boldsymbol{R}_q)\,.
\end{aligned}
\tag{5.29}
$$

Finally, we combine these results with the previous calculations for $E_{(2,2,2)}$ and obtain

$$
\begin{aligned}
E_{(2,2,2)}(\boldsymbol{R}_q) &= E_{(0,2,0)}(\boldsymbol{R}_q) - E_{(0,1,0)}(\boldsymbol{R}_q) + E_{(1,1,1)}(\boldsymbol{R}_q) - E_{(1,0,1)}(\boldsymbol{R}_q) + E_{(2,0,2)}(\boldsymbol{R}_q) \\
&= E_{(0,2,0)}(\boldsymbol{R}_q) - E_{(0,1,0)}(\boldsymbol{R}_q) \\
&\quad + \big[E_{(1,1,0)}(\boldsymbol{R}_q) - E_{(0,1,0)}(\boldsymbol{R}_q) + E_{(0,1,1)}(\boldsymbol{R}_q)\big] \\
&\quad - \big[E_{(1,0,0)}(\boldsymbol{R}_q) - E_{(0,0,0)}(\boldsymbol{R}_q) + E_{(0,0,1)}(\boldsymbol{R}_q)\big] \\
&\quad + \big[\big(E_{(2,0,0)}(\boldsymbol{R}_q) - E_{(1,0,0)}(\boldsymbol{R}_q)\big) + \big(E_{(1,0,1)}(\boldsymbol{R}_q) - E_{(0,0,1)}(\boldsymbol{R}_q)\big) + E_{(0,0,2)}(\boldsymbol{R}_q)\big] \\
&= E_{(0,2,0)}(\boldsymbol{R}_q) - 2E_{(0,1,0)}(\boldsymbol{R}_q) + E_{(1,1,0)}(\boldsymbol{R}_q) + E_{(0,1,1)}(\boldsymbol{R}_q) - 2E_{(1,0,0)}(\boldsymbol{R}_q) \\
&\quad + E_{(0,0,0)}(\boldsymbol{R}_q) - 2E_{(0,0,1)}(\boldsymbol{R}_q) + E_{(2,0,0)}(\boldsymbol{R}_q) + E_{(1,0,1)}(\boldsymbol{R}_q) + E_{(0,0,2)}(\boldsymbol{R}_q)
\end{aligned}
\tag{5.30}
$$

This is exactly the spelled out version of equation 5.17 for $E_{(2,2,2)}$.

As discussed in Section 5.3.5, practical considerations motivate us to modify the machine learning model $E_{(2,2,2)}$ to further include the training data from subspace $(2, 2, 0)$, i.e. data from the most expensive subspace with CCSD(T) calculations on a cc-pvdz basis set. Therefore, we introduced the *shifted* machine learning model $E^{shifted}_{(2,2,2)}$ in Section 5.3.5. In principle, it can be computed by using the modified index set $\mathcal{I}_{shifted}$ together with equations 5.15 and 5.16. However, we can also recursively derive the shifted model by first computing the machine learning model $E_{(4,4,4)}$, which, derived similar to $E_{(2,2,2)}$, becomes

$$
\begin{aligned}
E_{(4,4,4)}(\boldsymbol{R}_q) = \\
&E_{(0,4,0)}(\boldsymbol{R}_q) + E_{(0,3,1)}(\boldsymbol{R}_q) + E_{(1,3,0)}(\boldsymbol{R}_q) + E_{(0,2,2)}(\boldsymbol{R}_q) + E_{(1,2,1)}(\boldsymbol{R}_q) + E_{(2,2,0)}(\boldsymbol{R}_q) \\
&+ E_{(0,1,3)}(\boldsymbol{R}_q) + E_{(1,1,2)}(\boldsymbol{R}_q) + E_{(2,1,1)}(\boldsymbol{R}_q) + E_{(3,1,0)}(\boldsymbol{R}_q) + E_{(0,0,4)}(\boldsymbol{R}_q) + E_{(1,0,3)}(\boldsymbol{R}_q) \\
&+ E_{(2,0,2)}(\boldsymbol{R}_q) + E_{(3,0,1)}(\boldsymbol{R}_q) + E_{(4,0,0)}(\boldsymbol{R}_q) - 2E_{(0,3,0)}(\boldsymbol{R}_q) - 2E_{(0,2,1)}(\boldsymbol{R}_q)
\end{aligned}
$$

$$
\begin{aligned}
&- 2E_{(1,2,0)}(\boldsymbol{R}_q) - 2E_{(0,1,2)}(\boldsymbol{R}_q) - 2E_{(1,1,1)}(\boldsymbol{R}_q) - 2E_{(2,1,0)}(\boldsymbol{R}_q) - 2E_{(0,0,3)}(\boldsymbol{R}_q) \\
&- 2E_{(1,0,2)}(\boldsymbol{R}_q) - 2E_{(2,0,1)}(\boldsymbol{R}_q) - 2E_{(3,0,0)}(\boldsymbol{R}_q) + E_{(0,2,0)}(\boldsymbol{R}_q) + E_{(0,1,1)}(\boldsymbol{R}_q) \\
&+ E_{(1,1,0)}(\boldsymbol{R}_q) + E_{(0,0,2)}(\boldsymbol{R}_q) + E_{(1,0,1)}(\boldsymbol{R}_q) + E_{(2,0,0)}(\boldsymbol{R}_q)
\end{aligned}
$$

In a second step, we exclude those subspaces, that are not contained in $\mathcal{I}_{shifted}$, i.e. the subspaces for the multi-indices

$$(0,4,0), (0,3,1), (1,3,0), (3,1,0), (3,0,1), (4,0,0), (0,3,0), (3,0,0)\,. \tag{5.31}$$

For symmetry reasons and to keep a valid combination technique rule for arbitrary index sets, we also have to exclude one contribution of the subspaces with multi-indices

$$(2,0,0), (2,1,0), (2,0,1)\,. \tag{5.32}$$

The resulting shifted machine learning model $E^{shifted}_{(2,2,2)}$ is given in Eq. 5.19.

Supporting information

Geometries are provided as xyz files. Two types of energy data are available for each of the three basis sets (sto-3g, 6-31g and cc-pvdz), i.e., the total energy (E) and effective averaged atomization energies (E^*). The latter is defined as $E - \sum_I n_I * e_I$, where n_I is the number of atom I in the molecule and e_I is the effective atomic energy of atom I obtained through a linear least square fit of $E = \sum_I n_I * e_I$ for all molecules in the dataset. Free atom energies for all basis sets and electron methods are also included. Every type of energy data for any basis set used is given as a text file, consisting of three columns representing HF, MP2 and CCSD(T) energies, respectively.

Acknowledgement

We are grateful for discussions with P. D. Mezei and M. Schwilk. This collaboration is mainly being funded by the Swiss National Science foundation through 407540_167186 NFP 75 Big Data. OAvL also acknowledges additional support by the Swiss National Science foundation (No. PP00P2_138932, 200021_175747, NCCR MARVEL). Some calculations were performed at sciCORE (http://scicore.unibas.ch/) scientific computing core facility at University of Basel.

References

[BDJTVL18] T. Bereau, R. A. DiStasio Jr, A. Tkatchenko, and O. A. Von Lilienfeld. Non-covalent interactions across organic and biological subsets of chemical space: Physics-based potentials parametrized from machine learning. *The Journal of Chemical Physics*, 148(24):241706, 2018.

[BDP+17a] A. P. Bartók, S. De, C. Poelking, N. Bernstein, J. R. Kermode, G. Csányi, and

M. Ceriotti. Machine learning unifies the modeling of materials and molecules. *Science Advances*, 3(12), 2017.

[BDP+17b] A. P. Bartók, S. De, C. Poelking, N. Bernstein, J. R. Kermode, G. Csányi, and M. Ceriotti. Machine learning unifies the modeling of materials and molecules. *Science advances*, 3(12):e1701816, 2017.

[Beh11] J. Behler. Atom-centered symmetry functions for constructing high-dimensional neural network potentials. *J. Chem. Phys.*, 134(7):074106, 2011.

[BG04] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:147–269, 2004.

[BKC13] A. P. Bartók, R. Kondor, and G. Csányi. On representing chemical environments. *Phys. Rev. B*, 87:184115, May 2013.

[BOA+04] A. D. Boese, M. Oren, O. Atasoylu, J. M. L. Martin, M. Kállay, and J. Gauss. *J. Chem. Phys.*, 120:4129, 2004.

[BPKC10] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi. Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons. *Phys. Rev. Lett.*, 104:136403, Apr 2010.

[BRvLR17] N. J. Browning, R. Ramakrishnan, O. A. von Lilienfeld, and U. Roethlisberger. Genetic optimization of training sets for improved machine learning models of molecular properties. *J. Phys. Chem. Lett.*, 8(7):1351, 2017.

[Ced98] G. Ceder. Predicting properties from scratch. *Science*, 280(5366):1099–1100, 1998.

[CFH+17] A. S. Christensen, F. A. Faber, B. Huang, L. A. Bratholm, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld. Qml: A python toolkit for quantum machine learning, 2017.

[CGH18] S. R. Chinnamsetty, M. Griebel, and J. Hamaekers. An adaptive multiscale approach for electronic structure methods. *Multiscale Modeling & Simulation*, 16(2):752–776, 2018.

[CJS+94] C. Cortes, L. D. Jackel, S. A. Solla, V. Vapnik, and J. S. Denker. Learning curves: Asymptotic values and rate of convergence. In *Advances in Neural Information Processing Systems*, pages 327–334, 1994.

[CMSY12] A. J. Cohen, P. Mori-Sánchez, and W. Yang. Challenges for density functional theory. *Chem. Rev.*, 112(1):289–320, 2012. PMID: 22191548.

[CRR+99] L. A. Curtiss, P. C. Redfern, K. Raghavachari, V. Rassolov, and J. A. Pople. Gaussian-3 theory using reduced moøller-plesset order. *The Journal of chemical physics*, 110(10):4703–4709, 1999.

[CRRP00] L. A. Curtiss, K. Raghavachari, P. C. Redfern, and J. A. Pople. Gaussian-3 theory using scaled energies. *J. Chem. Phys.*, 112(3):1125–1132, 2000.

[CRTP91a] L. A. Curtiss, K. Raghavachari, G. W. Trucks, and J. A. Pople. Gaussian-2 theory for molecular energies of first-and second-row compounds. *J. Chem. Phys.*, 94(11):7221–7230, 1991.

[CRTP91b] L. A. Curtiss, K. Raghavachari, G. W. Trucks, and J. A. Pople. Gaussian-2 theory for molecular energies of first-and second-row compounds. *The Journal of chemical physics*, 94(11):7221–7230, 1991.

[CTS+17] S. Chmiela, A. Tkatchenko, H. E. Sauceda, I. Poltavsky, K. T. Schütt, and K.-R. Müller. Machine learning of accurate energy-conserving molecular force fields. *Sci. Adv.*, 3(5):e1603015, 2017.

[DBCC16] S. De, A. P. Bartok, G. Csanyi, and M. Ceriotti. Comparing molecules and solids across structural and alchemical space. *Phys. Chem. Chem. Phys.*, 18:13754–13769, 2016.

[DOYT17] P. O. Dral, A. Owens, S. N. Yurchenko, and W. Thiel. Structure-based sampling and self-correcting machine learning for accurate calculations of potential energy surfaces and vibrational levels. *The Journal of Chemical Physics*, 146(24):244108, 2017.

[FBR05] T. Fink, H. Bruggesser, and J.-L. Reymond. Virtual exploration of the small-molecule chemical universe below 160 daltons. 44(10):1504–1508, 2005.

[FCHvL18] F. A. Faber, A. S. Christensen, B. Huang, and O. A. von Lilienfeld. Alchemical and structural distribution based representation for universal quantum machine learning. *The Journal of Chemical Physics*, 148(24):241717, 2018.

[FHH+17] F. A. Faber, L. Hutchison, B. Huang, J. Gilmer, S. S. Schoenholz, G. E. Dahl, O. Vinyals, S. Kearnes, P. F. Riley, and O. A. von Lilienfeld. Prediction errors of molecular machine learning models lower than hybrid DFT error. *J. Chem. Theory Comput.*, 13:5255–5264, 2017.

[FLvLA15] F. Faber, A. Lindmaa, O. A. von Lilienfeld, and R. Armiento. Crystal structure representations for machine learning models of formation energies. *Int. J. Quantum Chem.*, 115:1094, 2015. `http://arxiv.org/abs/1503.07406`.

[FLvLA16] F. A. Faber, A. Lindmaa, O. A. von Lilienfeld, and R. Armiento. Machine learning energies of 2 million elpasolite (abC_2D_6) crystals. *Phys. Rev. Lett.*, 117:135502, Sep 2016.

[FTS+] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, J. A. Montgomery, Jr., T. Vreven, K. N. Kudin, J. C. Burant, J. M. Millam, S. S. Iyengar, J. Tomasi, V. Barone, B. Mennucci, M. Cossi, G. Scalmani, N. Rega, G. A. Petersson, H. Nakatsuji, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, M. Klene, X. Li, J. E. Knox, H. P. Hratchian, J. B. Cross, C. Adamo, J. Jaramillo, R. Gomperst, R. E. Stratmann, O. Yazyev, A. J.

Austin, R. Cammi, C. Pomelli, J. W. Ochterski, P. Y. Ayala, K. Morokuma, G. A. Voth, P. Salvador, J. J. Dannenberg, V. G. Zakrzewski, J. B. Foresman, J. V. Ortiz, Q. Cui, A. G. Baboul, S. Clifford, J. Cioslowski, B. B. Stefanov, G. Liu, A. Liashenko, P. Piskorz, I. Komaromi, R. L. Martin, D. J. Fox, T. Keith, M. A. Al-Laham, C. Y. Peng, A. Nanayakkara, M. Challacombe, P. M. W. Gill, B. Johnson, W. Chen, M. W. Wong, C. Gonzalez, and J. A. Pople. Gaussian 09. Gaussian Inc. Wallingford CT 2009.

[GBM17] M. Gastegger, J. Behler, and P. Marquetand. Machine learning molecular dynamics for the simulation of infrared spectra. *Chemical science*, 8(10):6924–6935, 2017.

[GH13a] M. Griebel and H. Harbrecht. A note on the construction of L-fold sparse tensor product spaces. *Constructive Approximation*, 38(2):235–251, 2013.

[GH13b] M. Griebel and H. Harbrecht. On the construction of sparse tensor product spaces. *Mathematics of Computation*, 82(282):975–994, 2013.

[GH14] M. Griebel and H. Harbrecht. On the convergence of the combination technique, in: *Lect. Notes Comput. Sci. Eng.: "Sparse grids and applications—Munich 2012"*, springer, cham. 97:55–74, 2014.

[GHH11] M. Griebel, J. Hamaekers, and F. Heber. Bossanova: A bond order dissection approach for efficient electronic structure calculations. *Oberwolfach Report*, 32:1804–1808, 2011.

[GHH14] M. Griebel, J. Hamaekers, and F. Heber. A bond order dissection anova approach for efficient electronic structure calculations. in: *Extraction of Quantifiable Information from Complex Systems*, springer. pages 211–235, 2014.

[GK00] M. Griebel and S. Knapek. Optimized tensor-product approximation spaces. *Constructive Approximation. An International Journal for Approximations and Expansions*, 16(4):525–540, 2000.

[GK09] M. Griebel and S. Knapek. Optimized general sparse grid approximation spaces for operator equations. *Mathematics of Computation*, 78(268):2223–2257, 2009.

[GPS18] K. Gubaev, E. V. Podryabinkin, and A. V. Shapeev. Machine learning of molecular properties: Locality and active learning. *The Journal of Chemical Physics*, 148(24):241727, 2018.

[GSR+17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, 2017.

[GSZ92] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems, in: *Iterative methods in linear algebra (Brussels, 1991)*, north-holland, amsterdam. pages 263–281, 1992.

[GVMVDS18] M. M. Ghahremanpour, P. J. Van Maaren, and D. Van Der Spoel. The alexandria library, a quantum-chemical database of molecular properties for force field development. *Scientific data*, 5:180062, 2018.

[HGC07] M. Hegland, J. Garcke, and V. Challis. The combination technique and some generalisations. *Linear Algebra and its Applications*, 420(2-3):249–275, 2007.

[HJO00] T. Helgaker, P. Jørgensen, and J. Olsen. *Molecular Electronic-Structure Theory*. John Wiley & Sons, LTD, 2000.

[HMB+13] K. Hansen, G. Montavon, F. Biegler, S. Fazli, M. Rupp, M. Scheffler, O. A. von Lilienfeld, A. Tkatchenko, and K.-R. Müller. Assessment and validation of machine learning methods for predicting molecular atomization energies. *J. Chem. Theory Comput.*, 9(8):3404–3419, 2013.

[HPS13a] H. Harbrecht, M. Peters, and M. Siebenmorgen. Combination technique based k-th moment analysis of elliptic problems with random diffusion. *Journal of Computational Physics*, 252:128–141, 2013.

[HPS13b] H. Harbrecht, M. Peters, and M. Siebenmorgen. On multilevel quadrature for elliptic stochastic partial differential equations, in: *Lect. Notes Comput. Sci. Eng.: "Sparse grids and applications"*, springer, heidelberg. 88:161–179, 2013.

[HR17] H. Huo and M. Rupp. Unified representation for machine learning of molecules and crystals. *arXiv preprint arXiv:1704.06439*, 2017.

[HSL18] B. Huang, N. O. Symonds, and O. A. v. Lilienfeld. Quantum machine learning in chemistry and materials. *Handbook of Materials Modeling: Methods: Theory and Modeling*, pages 1–27, 2018.

[HvL16] B. Huang and O. A. von Lilienfeld. Communication: Understanding molecular representations in machine learning: The role of uniqueness and target similarity. *J. Chem. Phys.*, 145(16):161102, 2016.

[HvL17a] B. Huang and O. von Lilienfeld. The dna of chemistry: Scalable quantum machine learning with amons. *arXiv:1707.04146*, 2017.

[HvL17b] B. Huang and O. A. von Lilienfeld. The DNA of chemistry: Scalable quantum machine learning with amons. *arXiv preprint arXiv:1707.04146*, 2017. submitted to Nature.

[HWCE06] J. Hafner, C. Wolverton, G. Ceder, and G. Editors. Toward computational materials design: The impact of density functional theory on materials research. *MRS Bulletin*, 31:659, 2006.

[JK17] J. P. Janet and H. J. Kulik. Predicting electronic structure properties of transition metal complexes with neural networks. *Chemical Science*, 8(7):5137–5152, 2017.

[Kar90] M. Karplus. Three-dimensional "pople diagram". *Journal of Physical Chemistry*, 94(14):5435–5436, 1990.

[Kar16a] A. Karton. A computational chemist's guide to accurate thermochemistry for organic molecules. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 6(3):292–310, 2016.

[Kar16b] A. Karton. A computational chemist's guide to accurate thermochemistry for organic molecules. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 6(3):292–310, 2016.

[KCY+15] C. Kefalidis, L. Castro, A. Yahia, L. Perrin, and L. Maron. Computational methods in lanthanide and actinide chemistry, ed. m. dolg, 2015.

[KE04] P. Kirkpatrick and C. Ellis. Chemical space. *Nature*, 432:823, 2004.

[KH15] W. Koch and M. C. Holthausen. *A chemist's guide to density functional theory.* John Wiley & Sons, 2015.

[KO00] M. C. Kennedy and A. O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 2000.

[MFM+96] K. R. Müller, M. Finke, N. Murata, K. Schulten, and S. Amari. A numerical study on learning curves in stochastic multilayer feedforward networks. *Neural Comp.*, 8:1085, 1996.

[MJFOP99] J. A. Montgomery Jr, M. J. Frisch, J. W. Ochterski, and G. A. Petersson. A complete basis set model chemistry. vi. use of density functional geometries and frequencies. *J. Chem. Phys.*, 110(6):2822–2827, 1999.

[MJFOP00] J. A. Montgomery Jr, M. J. Frisch, J. W. Ochterski, and G. A. Petersson. A complete basis set model chemistry. vii. use of the minimum population localization method. *J. Chem. Phys.*, 112(15):6532–6542, 2000.

[MO99] J. M. L. Martin and G. d. Oliveira. *J. Chem. Phys.*, 111:1843, 1999.

[MOP] MOPAC2009, James J. P. Stewart, Stewart Computational Chemistry, Colorado Springs, CO, USA, HTTP://OpenMOPAC.net (2008).

[MRG+13a] G. Montavon, M. Rupp, V. Gobre, A. Vazquez-Mayagoitia, K. Hansen, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld. Machine learning of molecular electronic properties in chemical compound space. *New Journal of Physics*, 15(9):095003, 2013.

[MRG+13b] G. Montavon, M. Rupp, V. Gobre, A. Vazquez-Mayagoitia, K. Hansen, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld. Machine learning of molecular electronic properties in chemical compound space. *New Journal of Physics*, 15(9):095003, 2013.

[Mul17] A. Mullard. The drug-maker's guide to the galaxy. *Nature News*, 549(7673):445, 2017.

[OPMJ96] J. W. Ochterski, G. A. Petersson, and J. A. Montgomery Jr. A complete basis set model chemistry. v. extensions to six or more heavy atoms. *J. Chem. Phys.*, 104(7):2598–2619, 1996.

[Pfl97] C. Pflaum. Convergence of the combination technique for second-order elliptic differential equations. *SIAM Journal on Numerical Analysis*, 34(6):2431–2455, 1997.

[PGL17] G. Pilania, J. E. Gubernatis, and T. Lookman. Multi-fidelity machine learning models for accurate bandgap predictions of solids. *Computational Materials Science*, 129:156–163, 2017.

[PKLAG15] E. O. Pyzer-Knapp, K. Li, and A. Aspuru-Guzik. Learning from the harvard clean energy project: The use of neural networks to accelerate materials discovery. *Advanced Functional Materials*, 25(41):6495–6502, 2015.

[Pop65] J. Pople. Two-dimensional chart of quantum chemistry. *The Journal of Chemical Physics*, 43(10):S229–S230, 1965.

[Pop99] J. A. Pople. Nobel lecture: Quantum chemical models. *Reviews of Modern Physics*, 71(5):1267, 1999.

[PWJ+13] G. Pilania, C. Wang, X. Jiang, S. Rajasekaran, and R. Ramprasad. Accelerating materials property predictions using machine learning. *Scientific reports*, 3:2810, 2013.

[RDRvL14a] R. Ramakrishnan, P. Dral, M. Rupp, and O. A. von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1:140022, 2014.

[RDRvL14b] R. Ramakrishnan, P. Dral, M. Rupp, and O. A. von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Sci. Data*, 1:140022, 2014.

[RDRvL15] R. Ramakrishnan, P. Dral, M. Rupp, and O. A. von Lilienfeld. Big Data meets Quantum Chemistry Approximations: The Δ-Machine Learning Approach. *J. Chem. Theory Comput.*, 11:2087, 2015.

[Rei13] C. Reisinger. Analysis of linear difference schemes in the sparse grid combination technique. *IMA Journal of Numerical Analysis*, 33(2):544–581, 2013.

[RG18] A. Rüttgers and M. Griebel. Multiscale simulation of polymeric fluids using the sparse grid combination technique. *Applied Mathematics and Computation*, 319:425–443, 2018.

[RHTvL15] R. Ramakrishnan, M. Hartmann, E. Tapavicza, and O. A. von Lilienfeld. Electronic spectra from tddft and machine learning in chemical space. *The Journal of chemical physics*, 143(8):084111, 2015.

[RRvL15a] M. Rupp, R. Ramakrishnan, and O. A. von Lilienfeld. Machine learning for quantum mechanical properties of atoms in molecules. *J. Phys. Chem. Lett.*, 6:3309, 2015. `http://arxiv/abs/1505.00350`.

[RRvL15b] M. Rupp, R. Ramakrishnan, and O. A. von Lilienfeld. Machine learning for quantum mechanical properties of atoms in molecules. *The Journal of Physical Chemistry Letters*, 6(16):3309–3313, 2015.

[RTMvL12a] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Phys. Rev. Lett.*, 108:058301, 2012.

[RTMvL12b] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Phys. Rev. Lett.*, 108(5):058301, Jan 2012.

[RvDBR12] L. Ruddigkeit, R. van Deursen, L. Blum, and J.-L. Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *J. Chem. Inf. Model.*, 52:2684, 2012.

[RvL17] R. Ramakrishnan and O. A. von Lilienfeld. *Machine Learning, Quantum Chemistry, and Chemical Space*, volume 30, pages 225–256. John Wiley & Sons, Inc., 2017.

[RvLB18] M. Rupp, O. A. von Lilienfeld, and K. Burke. Guest editorial: Special topic on data-enabled theoretical chemistry. *The Journal of Chemical Physics*, 148(24):241401, 2018.

[RW06] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning,* `www.GaussianProcess.org`. MIT Press, Cambridge, 2006. Editor: T. Dietterich.

[SAC+17] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nat. Comm.*, 8:13890, 2017.

[SC18] G. Schmitz and O. Christiansen. Gaussian process regression to accelerate geometry optimizations relying on numerical differentiation. *The Journal of Chemical Physics*, 148(24):241704, 2018.

[SIR17] J. S. Smith, O. Isayev, and A. E. Roitberg. Ani-1, a data set of 20 million calculated off-equilibrium conformations for organic molecules. *Scientific data*, 4:170193, 2017.

[SNL+18] J. S. Smith, B. Nebgen, N. Lubbers, O. Isayev, and A. E. Roitberg. Less is more: Sampling chemical space with active learning. *The Journal of Chemical Physics*, 148(24):241733, 2018.

[SNZ+18] J. S. Smith, B. T. Nebgen, R. Zubatyuk, N. Lubbers, C. Devereux, K. Barros, S. Tretiak, O. Isayev, and A. Roitberg. Outsmarting Quantum Chemistry Through Transfer Learning. 7 2018.

[SR18] G. N. Simm and M. Reiher. Error-controlled exploration of chemical reaction networks with gaussian processes. *arXiv preprint arXiv:1805.09886*, 2018.

[SSK+18] K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller. Schnet–a deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, 148(24):241722, 2018.

[SY18] L. Shen and W. Yang. Molecular dynamics simulations with quantum mechanics/molecular mechanics and adaptive neural networks. *Journal of chemical theory and computation*, 14(3):1442–1455, 2018.

[Vap13] V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[vL13] O. A. von Lilienfeld. First principles view on chemical compound space: Gaining rigorous atomistic control of molecular properties. *International Journal of Quantum Chemistry*, 113(12):1676–1689, 2013.

[vL14] O. A. von Lilienfeld. *Towards the Computational Design of Compounds from First Principles*, volume IX of *Mathematical Physics Studies*. Springer, 2014.

[vL18] O. A. von Lilienfeld. Quantum machine learning in chemical compound space. *Angewandte Chemie International Edition*, 2018. http://dx.doi.org/10.1002/anie.201709686.

[WKK+15] H.-J. Werner, P. J. Knowles, G. Knizia, F. R. Manby, M. Schütz, P. Celani, W. Györffy, D. Kats, T. Korona, R. Lindh, A. Mitrushenkov, G. Rauhut, K. R. Shamasundar, T. B. Adler, R. D. Amos, A. Bernhardsson, A. Berning, D. L. Cooper, M. J. O. Deegan, A. J. Dobbyn, F. Eckert, E. Goll, C. Hampel, A. Hesselmann, G. Hetzer, T. Hrenar, G. Jansen, C. Köppl, Y. Liu, A. W. Lloyd, R. A. Mata, A. J. May, S. J. McNicholas, W. Meyer, M. E. Mura, A. Nicklass, D. P. O'Neill, P. Palmieri, D. Peng, K. Pflüger, R. Pitzer, M. Reiher, T. Shiozaki, H. Stoll, A. J. Stone, R. Tarroni, T. Thorsteinsson, and M. Wang. Molpro, version 2015.1, a package of ab initio programs, 2015.

[WMC18] M. J. Willatt, F. Musil, and M. Ceriotti. A data-driven construction of the periodic table of the elements. *arXiv preprint arXiv:1807.00236*, 2018.

6 Algorithmic patterns for $\mathcal{H}$ matrices on many-core processors

6.1 Introduction

In many fields of applications we are required to solve large dense systems of linear equations of the form

$$A_{\phi,\mathcal{Y}\times\mathcal{Y}}\boldsymbol{x} = \boldsymbol{b} \tag{6.1}$$

with

$$A_{\phi,\mathcal{Y}\times\mathcal{Y}} = \begin{pmatrix} \phi(\boldsymbol{y}_1,\boldsymbol{y}_1) & \cdots & \phi(\boldsymbol{y}_1,\boldsymbol{y}_N) \\ \vdots & \ddots & \vdots \\ \phi(\boldsymbol{y}_N,\boldsymbol{y}_1) & \cdots & \phi(\boldsymbol{y}_N,\boldsymbol{y}_N) \end{pmatrix}, \quad \boldsymbol{x},\boldsymbol{b} \in \mathbb{R}^N . \tag{6.2}$$

where $\mathcal{Y} := \{\boldsymbol{y}_1,\ldots,\boldsymbol{y}_N\} \subset \Omega$ is a set of N points in a space $\Omega \subset \mathbb{R}^d$ and $\phi : \Omega \times \Omega \to \mathbb{R}$ is a bivariate *kernel* function operating on that domain. In *kernel-based interpolation* [Wen04], the linear system (6.1) arises in the computation of interpolation coefficients. In *Gaussian Process Regression* (GPR) [RW05], kernel ϕ is a covariance function and $A_{\phi,\mathcal{Y}\times\mathcal{Y}}$ is replaced by $(A_{\phi,\mathcal{Y}\times\mathcal{Y}} + \sigma^2 I)$ with σ^2 a (scalar) variance and I the unity matrix. The same modified system also shows up in *kernel ridge regression* [Vov13]. Integral equations, discretized by e.g. collocation, or discretizations by the boundary element method (BEM) lead to similar linear systems.

The problem size N might get very large. As an example, N could be the number of training samples in machine learning by kernel ridge regression or the number of degrees of freedom in BEM. This can grow up to tens to hundreds of millions of samples or even more, depending on the application. Here, linear solvers for (6.1) based on direct factorization get intractable due to their $O(N^3)$ complexity. This is overcome by iterative solvers with fast approximate dense matrix-vector product.

In this work, we address the topic of parallelizing the fast approximate dense matrix-vector product based on *hierarchical matrices ($\mathcal{H}$ matrices)* [BGH03, Beb08, Hac15, Hac16] in context of the model problem in (6.1–6.2).[1] Using $\mathcal{H}$ matrix techniques, a matrix-vector product for a fixed approximation accuracy is done in $O(N \log N)$ operations, given ϕ is *asymptotically smooth*, cf. Section 6.2. Similar to panel clustering [HN89] and multipole techniques [GR97], the core idea is to distinguish between subsets $\mathcal{Y}_i \times \mathcal{Y}_j \subset \mathcal{Y} \times \mathcal{Y}$, where $\mathcal{Y}_i$ and $\mathcal{Y}_j$ are "close" to each other or "far away". In $\mathcal{H}$ matrices, a *tree-based* spatial decomposition of $\mathcal{Y} \times \mathcal{Y}$ is done.

[1] In the following, we stick to model problem (6.1) with collocation matrices of type (6.2), where the evaluation of each matrix entry is cheap. In contrast, the evaluation of system matrix entries in e.g. the discretization by the BEM approach is much more expensive. That is, we here focus on the implementation and optimization of the (many-core) parallel $\mathcal{H}$ matrix *algorithms* instead of the optimization of the performance of the *evaluation* of the kernel matrix entries, which is a different objective, cf. [BC15].

Nodes in that tree correspond to subsets of $\mathcal{Y}_i \times \mathcal{Y}_j \subset \mathcal{Y} \times \mathcal{Y}$ and thus to sub-blocks of $A_{\phi,\mathcal{Y}\times\mathcal{Y}}$. Based on an *admissibility condition*, these sub-blocks are either identified as close and thus directly evaluated or as far away and thus approximated. Approximation is done either using expansions of the kernel function ϕ or using low-rank approximations of the algebraically given matrix sub-block. In this work, low-rank approximations by adaptive cross approximation (ACA) [BR03] are considered. This leads to a purely algebraic approach. A further refinement of $\mathcal{H}$ matrix techniques are $\mathcal{H}^2$ matrices [HKS00, HB02, Bör04] that even exhibit $O(N)$ time complexity. Nevertheless, due to a higher algorithmic complexity, we for now stick to the classical $\mathcal{H}$ matrix techniques.

$\mathcal{H}$ matrix techniques speed up the solution process of (6.1) significantly. Nevertheless, large to huge problem sizes still cannot be tackled using a single processor core or just one workstation with a limited amount of memory. Therefore parallelization of the $\mathcal{H}$ matrix method is crucial. Parallelization of $\mathcal{H}$ matrix methods on *standard processors* (CPUs) is an active research field. Research in this domain ranges from shared-memory to distributed-memory parallel $\mathcal{H}$ matrix implementations on CPUs. The results of this research are a set of parallel $\mathcal{H}$ matrix libraries, which include, but are not limited to $\mathcal{H}$-`Lib`$^{\texttt{pro}}$ [Kri17, BGH03, Kri05, GKLB08], which is rather feature-complete with a shared-memory parallelization and limited distributed-memory support, `AHMED` (Another software library on hierarchical matrices for elliptic differential equations) [Beb], `DMHM` (Distributed-Memory Hierarchical Matrices) [Pou] with a distributed-memory parallelization, `H2Lib` [BÏ7] with support for shared-memory parallelism and work based on the related Hierarchically Semi-Separable (HSS) matrices [SDC07] with the software `STRUMPACK` [RLGN16, GLR$^+$16], where the latter one is parallelized for shared- and distributed-memory. Another related, strongly CPU-parallel software for problems of type (6.1), (6.2) is PetRBF [YBK10]. In contrast to the above works, we here address parallelization on *many-core processors*.

Many-core processors such as graphics processing units (GPUs) or Intel Xeon Phi reflect recent developments in chip production and high performance computing (HPC): Future parallel computers might show a dramatic growth in the number of parallel processing units with a strong (negative) impact on scalability of current shared-memory and distributed-memory parallelizations. Many-core processors are often assumed to be an optimal testbed for reformulations of classical algorithms towards a massive amount of parallelism, preparing for future parallel computers.

In this work, we will discuss fundamental research on new formulations of standard $\mathcal{H}$ matrix algorithms in order to expose as much parallelism as possible to many-core hardware. Our new algorithms are then implemented on a model hardware, namely GPUs (by *NVIDIA*). We claim that all of our algorithmic developments equivalently apply to GPU hardware of other vendors or to the Xeon Phi architecture. There is a small set of related work for $\mathcal{H}$ matrices on many-core hardware. In [BC15], the GPU-acceleration of the quadrature in a $\mathcal{H}^2$ matrix method for boundary element method problems is considered. Moreover, in [Kri13], many-core parallel LU-factorization for $\mathcal{H}$ matrices is presented and evaluated on a Xeon Phi device. However, these works have in common that many-core hardware is only used as an *accelerator* or for another computing task, and not as main computing device for the fast matrix-vector product. In contrast, we want to rely completely on many-core parallel hardware for the full $\mathcal{H}$ matrix construction and matrix-vector product.

Other works in the field of many-core hardware concentrating on matrices of type (6.2) or using other methods are the `ASKIT` library [MXYB16], which uses GPU acceleration and some very specific tree-based approximation technique and fast multipole methods [YB13, ABC+14] with e.g. the multi-GPU parallel library `ExaFMM` [YB13]. While these approaches are very promising for these specific matrices, our main intention is to parallelize the entirely algebraic $\mathcal{H}$ matrix technique, allowing it to be used in much more applications.

Fully relying on many-core hardware specifically requires us to provide many-core parallel reformulations of the underlying spatial data structure, the tree construction and traversal, bounding box computations and the construction and evaluation of both the dense matrix parts as well as the low-rank matrix approximations. We propose several algorithmic patterns for many-core processors in context of $\mathcal{H}$ matrices. Space filling curves, i.e. Z order curves, are discussed as parallelized spatial data structure. This goes back to work on the fast construction and evaluation of bounding volume hierarchies on GPUs [LGS+09]. We use a parallel formulation of tree traversal using an array-based tree description (cf. [MGG12] for a background on GPU-based tree traversal). Batching or work aggregation, cf. e.g. [CKL, AHTD17] allows to express parallelism even for code parts in which many similar non-equally sized subtasks are done. This allows for strong optimizations of dense matrix operations, low-rank approximations and bounding box calculations. Note that we also address the problem of limited memory availability on many-core processors by (optionally) recomputing all data in the numerical linear algebra part. That is, only meta data such as the block cluster tree has to be stored permanently.

As a result of these developments, the author provides an Open Source reference implementation on GPU, which is called `hmglib` [Zas18]. To the best of the authors knowledge, this is the first entirely GPU-based $\mathcal{H}$ matrix library of this kind. For completeness, we should state that there is ongoing research on multi-GPU parallel hierarchical matrices in a library called `KSPARSE` [BLL+], which is, however, not published and not available for download. Since very recently, there exists a preprint [BTLK17] of the authors of [BLL+], discussing the parallel, batched GPU-based implementation of matrix factorizations in context of hierarchical matrices. However, it does not become clear, whether the full algorithm (beyond the batched linear algebra) is performed on GPU. Moreover, the underlying code is not published. Therefore, we still claim that the proposed work is the first available entirely GPU-based $\mathcal{H}$ matrix method.

From a technical point of view, we will show that our many-core parallel model implementation on one GPU outperforms a classical multi-core parallel CPU-based $\mathcal{H}$ matrix library running on roughly equally priced hardware by a factor of 50 in the $\mathcal{H}$ matrix construction and by a factor of four for the $\mathcal{H}$ matrix-vector product for a discussed model problem with cheap kernel evaluations. Nevertheless, our main intention is to show the changes that are to be done to get an entirely many-core parallel implementation. This shall lead to a better understanding and preparation for future intrinsically extremely parallel computing hardware.

Section 6.2 introduces hierarchical matrices and adaptive cross approximation. Thereafter, Section 6.3 discusses a simplified programming model for many-core processors. This model allows to provide many-core parallel algorithms for $\mathcal{H}$ matrices in Section 6.4. Section 6.5 treats the reference GPU implementation covering an in-depth benchmark and empirical performance analysis including a comparison to a multi-core CPU code. Finally, Section 6.6 concludes this work by a short summary.

6.2 $\mathcal{H}$ matrix background

In the following, we will briefly summarize the necessary algorithmical and mathematical aspects of $\mathcal{H}$ matrices. This overview is partially based on [BGH03]. For further reading see e.g. [Hac15].

Let us start by identifying the points in $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\}$ by their index set $I := \{1, \ldots, N\}$. A single entry $\phi(\boldsymbol{y}_i, \boldsymbol{y}_j)$ of the system matrix $A_{\phi,\mathcal{Y}\times\mathcal{Y}}$ corresponds an index tuple (i, j). Later, we will build *clusters* τ, i.e. specific subsets $\tau \subset I$. We can identify the product of two clusters, e.g. $\tau \times \sigma \subset I \times I$, with a sub-matrix $A_{\phi,\mathcal{Y}\times\mathcal{Y}}|_{\tau\times\sigma}$ of the system matrix $A_{\phi,\mathcal{Y}\times\mathcal{Y}}$. We will need this dual view between sets of index tuples and matrix entries to better understand the basic algorithmic idea of $\mathcal{H}$ matrices.

A kernel function $\phi : \Omega \times \Omega \to \mathbb{R}$ is called *asymptotically smooth* if there are constants $C_{as1}, C_{as2} \in \mathbb{R}^{>0}$ such that

$$|\partial_x^{\boldsymbol{\alpha}} \partial_y^{\boldsymbol{\beta}} \phi(\boldsymbol{y}, \boldsymbol{y}')| \leqslant C_{as1}(C_{as2}\|\boldsymbol{y} - \boldsymbol{y}'\|)^{-|\boldsymbol{\alpha}|-|\boldsymbol{\beta}|}|\phi(\boldsymbol{y}, \boldsymbol{y}')|$$

for all $\boldsymbol{y}, \boldsymbol{y}' \in \Omega$ with $\boldsymbol{y} \neq \boldsymbol{y}'$ and all multi-indices $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{N}_0^d$. Fixing $\boldsymbol{y} \in \Omega$, the kernel evaluation $\phi(\boldsymbol{y}, \boldsymbol{y}_{far})$ of an approximately smooth kernel function can be approximated with a controlled, small error, in case the point $\boldsymbol{y}_{far} \in \Omega$ is *far away* from $\boldsymbol{y}$. In the $\mathcal{H}$ matrix approach, an *admissibility* condition identifies matrix blocks $A_{\phi,\mathcal{Y}\times\mathcal{Y}}|_{\tau\times\sigma}$ that represent interactions of points with indices τ that are *far away* from points with indices σ. Admissible matrix blocks are traditionally approximated via series expansions of the kernel ϕ. We here consider the well-known alternative approach to approximate the matrix blocks by algebraic low-rank approximations as e.g. in [BR03].

6.2.1 Clustering

The *cluster tree* $\mathcal{T}_I = (V_I, \gamma, \mu)$ is a hierarchical spatial data structure on I (or $\mathcal{Y}$). V_I is the set of nodes in the tree, γ a mapping $\gamma : V_I \to \mathcal{P}(V_I)$ of the nodes to their children and $\mu : V_I \to \mathcal{P}(I)$ a mapping of the nodes to their value. Here, the *value* of each node is a cluster in I, i.e. a subset of I. A cluster tree has to fulfill

(C1) $\mu(v) \in \mathcal{P}(I)\backslash\{\varnothing\}$, for all $v \in V_I$,

(C2) $\mu(\mathrm{root}(\mathcal{T})) = I$,

(C3) if $v \in V_I$ is a leaf, i.e. $\gamma(v) = \varnothing$, then $|\mu(v)| \leqslant C_{leaf}$ and

(C4) if $v \in V_I$ is no leaf, then it has exactly two sons $\gamma(v) = \{v_1, v_2\}$ and $\mu(v) = \mu(v_1) \cup \mu(v_2)$.

Thereby, the cluster tree divides the full set I (C2) into a hierarchy of clusters, where non-empty clusters of I (represented by nodes in $\mathcal{T}_I$, C1) are disjointly partitioned into two smaller clusters (C4). In case a cluster is no longer partitioned (thus represented by a leaf), its size is bounded from above by C_{leaf} (C3).

In cardinality-based clustering (CBC) [BGH03], an algorithm to create the cluster tree decomposes the sets $\tau = \gamma(v)$ such that the subsets in the child nodes of v have similar size. Moreover, the subsets shall build geometrically distinct clusters. A CBC based on space filling

Algorithm 8 Algorithm to build a block cluster tree

```
procedure BUILD_BLOCK_CLUSTER_TREE(v_1, v_2, w, C_leaf)
    (τ, σ) ← (μ(v_1), μ(v_2))
    if τ × σ is not admissible and |τ| > C_leaf and |σ| > C_leaf then
        γ(w) ← ∅
        for v'_1 ∈ γ(v_1) do        ▷ Loop over all combinations of children in both cluster trees.
            for v'_2 ∈ γ(v_2) do
                μ(w') ← μ(v'_1) × μ(v'_2)                ▷ Set block cluster of new node w'.
                γ(w) ← γ(w) ∪ {w'}                       ▷ Add new node to children of w.
                BUILD_BLOCK_CLUSTER_TREE(v'_1, v'_2, w', C_leaf)
    else
        γ(w) ← ∅                    ▷ No child nodes are created, i.e. w becomes a leaf.
```

curves will be introduced in Section 6.4.1. The splitting in the cluster tree construction is continued as long as $|\tau| > C_{leaf}$.

6.2.2 Bounding box admissibility

In this work, we will restrict ourselves to an admissibility condition based on bounding boxes for clusters. Other choices are possible [Hac15]. For a cluster $\tau \subset I$, the bounding box Q_τ is given as

$$Q_\tau := \prod_{i=1}^{d} \left[a_\tau^{(i)}, b_\tau^{(i)} \right]$$

with $a_\tau^{(i)} := \min_{j \in \tau} y_j^{(i)}$, $b_\tau^{(i)} := \max_{j \in \tau} y_j^{(i)}$ and $\boldsymbol{y}_j := \left(y_j^{(1)}, \ldots, y_j^{(d)} \right)^\top$. One possible admissibility condition for an index block $\tau \times \sigma \subset I \times I$ is

$$\min \{ \mathrm{diam}(Q_\tau), \mathrm{diam}(Q_\sigma) \} \leqslant \eta \mathrm{dist}(Q_\tau, Q_\sigma) \tag{6.3}$$

with $\eta \in \mathbb{R}^{>0}$ a parameter balancing convergence and algorithmic complexity. Diameter $\mathrm{diam}(Q_\tau)$ and distance $\mathrm{dist}(Q_\tau, Q_\sigma)$ of bounding boxes are defined by

$$\mathrm{diam}(Q_\tau) := \left(\sum_{i=1}^{d} (b_\tau^{(i)} - a_\tau^{(i)})^2 \right)^{1/2},$$

$$\mathrm{dist}(Q_\tau, Q_\sigma) := \left(\sum_{i=1}^{d} \left(\max \left\{ 0, b_\sigma^{(i)} - a_\tau^{(i)} \right\}^2 + \max \left\{ 0, b_\tau^{(i)} - a_\sigma^{(i)} \right\}^2 \right) \right)^{1/2}.$$

6.2.3 Block cluster tree

A hierarchy over *blocks* $\tau \times \sigma \subset I \times I$ is induced by the *block cluster tree* $\mathcal{T}_{I \times I} = (V_{I \times I}, \gamma, \mu)$, with γ the child node map and $\mu : V_{I \times I} \to \mathcal{P}(I \times I)$ the map of nodes to their values, i.e. blocks. Note that we re-use here the same notation (γ, μ) as for the cluster tree. Algorithm 16 implicitly

Algorithm 9 Adaptive cross approximation (ACA) for $A \in \mathbb{R}^{m \times n}$ [BK09][BR03]

function COMPUTE_ADAPTIVE_CROSS_APPROXIMATION(A, ϵ)
  $k_{max} \leftarrow k$
  for $r = 1, 2, \ldots, k$ **do**
    $\hat{\boldsymbol{u}}_r = A_{1:m,j_r} - \sum_{l=1}^{r-1} \boldsymbol{u}_l (\boldsymbol{v}_l)_{j_r}$, ▷ Col. index j_r depending on implementation
    $\boldsymbol{u}_r = (\hat{\boldsymbol{u}}_{i_r})^{-1} \hat{\boldsymbol{u}}_r$, with $|(\hat{\boldsymbol{u}}_r)_{i_r}| = \|\hat{\boldsymbol{u}}_r\|_\infty$ ▷ Row index i_r given as pivot position
    $\boldsymbol{v}_r = (A_{i_r,1:n})^\top - \sum_{l=1}^{r-1} (\boldsymbol{u}_l)_{i_r} \boldsymbol{v}_l$
    if $\left(\|\boldsymbol{u}_r\|_2 \|\boldsymbol{v}_r\|_2 \leqslant \frac{\epsilon(1.0-\eta)}{1.0+\epsilon} \left\| \sum_{l=1}^{r} \boldsymbol{u}_l \boldsymbol{v}_l \right\|_F \right)$ **then** ▷ Stopping criterion
      $k_{max} \leftarrow r$ ▷ k_{max} is adaptively found rank
      stop loop
  $U \leftarrow (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{k_{max}})$
  $V \leftarrow (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k_{max}})$
  return U, V

defines the block cluster tree. For given cluster tree nodes v_1, v_2 (corresponding to clusters $\mu(v_1) = \tau, \mu(v_2) = \sigma$), a block cluster tree node w with $\mu(w) := \mu(v_1) \times \mu(v_2)$ (corresponding to $\mu(w) = \tau \times \sigma$) and parameter C_{leaf}, this algorithm recursively constructs a block cluster tree. Procedure BUILD_BLOCK_CLUSTER_TREE is initially launched with v_1 and v_2 each being a root of the cluster tree $\mathcal{T}_I$ and node w is initialized to represent the index block $I \times I$. By construction, the leafs of $\mathcal{T}_{I \times I}$, namely $\mathcal{L}_{I \times I} := \{w \in V_{I \times I} \,|\, \gamma(w) = \varnothing\}$, correspond to index blocks that form a partition of $I \times I$.

6.2.4 Rk-matrices and adaptive cross approximation

If a node w in a block cluster tree corresponds to an index block $\tau \times \sigma \subset I \times I$ that is admissible, the corresponding sub-matrix $A_{\phi,\mathcal{Y} \times \mathcal{Y}}|_{\tau \times \sigma} \in \mathbb{R}^{|\tau| \times |\sigma|}$ is replaced by an $\mathbf{R}k$ *matrix* $R_{\tau \times \sigma} \in \mathbb{R}^{|\tau| \times |\sigma|}$. An $\mathbf{R}k$ matrix $R_{\tau \times \sigma}$ is given as

$$R_{\tau \times \sigma} = U_{\tau \times \sigma} V_{\tau \times \sigma}^\top, \quad U_{\tau \times \sigma} \in \mathbb{R}^{|\tau| \times k}, V_{\tau \times \sigma} \in \mathbb{R}^{|\sigma| \times k},$$

that is, it has a maximum rank of k. Moreover, using $U_{\tau \times \sigma}$ and $V_{\tau \times \sigma}$, a matrix-vector product involving $R_{\tau \times \sigma}$ can be computed in $O(r \cdot (|\tau| + |\sigma|))$ operations.

While there are many (problem-dependent) ways to approximate $A_{\phi,\mathcal{Y} \times \mathcal{Y}}|_{\tau \times \sigma}$, we here aim at using a purely algebraic low-rank approximation method to derive $R_{\tau \times \sigma}$. Our method of choice is the *adaptive cross approximation* (ACA) [BR03, BK09]. This method builds a low-rank approximation by an iterative rank-one update process that is terminated based on the error ϵ in the Frobenius norm $\|\cdot\|_F$.

One version of adaptive cross approximation is given in Algorithm 9. It follows the lines of [BK09]. The algorithm computes for a general matrix $A \in \mathbb{R}^{m \times n}$ and error threshold ϵ matrices $U \in \mathbb{R}^{m \times k_{max}}$, $V \in \mathbb{R}^{n \times k_{max}}$ such that $A \approx UV^\top$. If the algorithm terminates due to the stopping criterion, k_{max} becomes the (adaptively computed) rank such that the error $\|A - UV^\top\|_F$ is in the range of ϵ, while not being guaranteed to be strictly smaller than ϵ. Otherwise, the maximum rank of k_{max} is hit. Unfortunately, the choice of a column pivot index j_r is strongly problem-dependent. For simplicity, in the search for a new pivot element,

Algorithm 10 Matrix-vector product with an $\mathcal{H}$ matrix $L \in^{I \times I}$

```
function MATRIX_VECTOR_PRODUCT(L, w, x, z)
    if γ(w) ≠ ∅ then
        for w' ∈ γ(w) do
            MATRIX_VECTOR_PRODUCT(L, w', x, z)
    else
        τ × σ ← μ(w)
        if τ × σ is admissible then
            t ← V_{τ×σ}^⊤ x|_τ
            z|_τ ← z|_τ + U_{τ×σ} t
        else
            z|_τ ← z|_τ + L|_{τ×σ} x|_σ
    return z
```

we iteratively increase j_r until $\|\hat{\boldsymbol{u}}_r\|_2 > \epsilon_0$ for small ϵ_0 in the range of machine precision.[2] This corresponds to a choice made in [BR03, Section 3.1]. In our practical implementation, we will, however, avoid to evaluate the stopping criterion and will only impose the maximum rank k_{max}. As we will see in Section 6.5.4, k_{max} can be chosen rather small due to the exponential convergence of ACA for appropriate kernel functions ϕ. For more details on ACA, see [BK09, BR03].

6.2.5 $\mathcal{H}$-matrices and their matrix-vector product

Formally, a general matrix $L \in \mathbb{R}^{|I| \times |I|}$ is — for fixed $k \in \mathbb{N}$ and block cluster tree $\mathcal{T}_{I \times I}$ — called $\mathcal{H}$ *matrix* of block-wise rank k, if

$$\operatorname{rank}(L|_{\tau \times \sigma}) \leqslant k$$

for all index blocks $\tau \times \sigma$ in admissible leafs. The operation to transform an existing dense matrix, e.g. $A_{\phi,\mathcal{Y} \times \mathcal{Y}}$, to $\mathcal{H}$ matrix form is called *truncation*. It involves the introduction of a cluster tree $\mathcal{T}_I$, a block cluster tree $\mathcal{T}_{I \times I}$ and the computation of a low-rank approximation of matrix blocks corresponding to admissible leafs.

The (fast) matrix-vector product of an $\mathcal{H}$ matrix $L \in \mathbb{R}^{|I| \times |I|}$ with a vector $\boldsymbol{x} \in \mathbb{R}^{|I|}$, that is, the efficient evaluation of

$$\boldsymbol{z} := \boldsymbol{z} + L\boldsymbol{x}\,,$$

is summarized in Algorithm 10. The algorithm recursively traverses the block cluster tree for an initially given (root) node w and applies a low-rank matrix-vector product for admissible blocks and the full dense matrix for non-admissible blocks. If we launch MATRIX_VECTOR_PRODUCT with w corresponding to $I \times I$ and L being the truncated version of $A_{\phi,\mathcal{Y} \times \mathcal{Y}}$, it can be shown that the algorithm has a complexity of $O(k \cdot N \log N)$ [Hac15].

[2] Note that this choice may not be stable. However, in our practical experiments with the discussed model problems, we did not observe instabilities. Nevertheless, other applications might require a better, thus more expensive, heuristics for the choice of j_r.

6.3 Programming model for many-core parallel algorithms

In this Section, we introduce the terminology to describe efficient and scalable parallel many-core algorithms. Note that, to the best of the author's knowledge, a common abstract programming model for many-core architectures is still missing. Therefore, algorithmic work on GPUs or Xeon Phi often addresses many details of these architectures. In contrast, we use a strongly simplified programming model, avoiding most of the technical details of classical many-core literature. Our model is based on two observations. First, a crucial part of a lot of many-core parallel algorithms requires almost no interaction between the involved parallel compute units, that is, they are close to *embarrassingly* parallel. Second, vendors (or enthusiasts) provide extremely efficient many-core parallel implementations of base algorithms (reductions, scan operations, etc.) for more complex parallel algorithmic patterns. Therefore, we claim that we can build all algorithms of interest by combinations of almost embarrassingly parallel kernels and standardized parallel algorithms. They are defined in more detail in the following paragraphs.

6.3.1 Almost embarrassingly parallel kernels

The kind of compute kernels we discuss here are strongly related to the *bulk synchronous parallel model*, cf. [Val90]: We introduce an (in principle) infinite number of virtual parallel threads. In each parallel thread, the same piece of sequential code is executed. Different memory accesses / execution paths are realized by an index that is associated to each thread.

All threads are aggregated in a *kernel*, which gets the number of threads to execute at launch time. The kernel terminates when all threads have stopped the execution of the sequential code. The sequential code (per thread) can either use *local memory*, which can only be read by that single thread, or *global memory*, which is available to all threads. At the end of the kernel execution, all local memory data is lost while global memory entries remain available. Whenever a single thread writes to a given global memory entry, read or write operations on that memory entry (by another thread) are invalid / prohibited. Reading (without writing) from a common global memory location by multiple threads in one kernel is possible.

One exception to the write rule is available in case of *atomic* operations (usually ATOMIC_ADD or ATOMIC_COMPARE_AND_SWAP) on global memory. Atomic operations issued by different threads on one common global memory location are all correctly executed, even if this means that threads get serialized. However, the ordering of the execution is not assured. Therefore, atomic operations are only useful in very few cases (e.g. counters).

Note that the actual mapping of threads to hardware processing units is not part of the model. This especially allows to define parallel programs independent of the number of available hardware threads. Moreover, the beforehand given definition of computing kernels does not give any hints towards the performance of their actual mapping to a given hardware platform. Let us give some examples. In case of GPUs, global memory accesses are fast if they are done consecutively for consecutive thread indices, that is, threads 0,1,2,3,... access memory entries $e, e+1, e+2, e+3, \ldots$. In contrast, random access has rather low performance. Moreover, conditionals in the thread-sequential code of a kernel might have a severe impact on performance on GPUs if thread execution paths diverge within the vectorization on a multi-processor. On the Xeon Phi, conditionals might even have a more dramatic impact if they evaluate differently

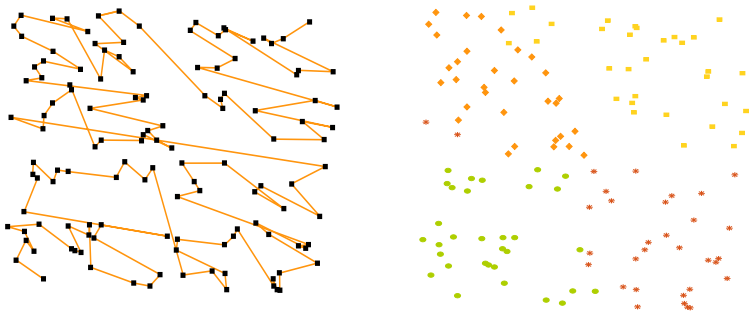


Figure 6.1: *Left:* By sorting a set of arbitrary points following their Morton codes, a spatial data structure is imposed. *Right:* By dividing the set of ordered points in to equally sized subsets, we implicitly create clusters.

within the less flexible AVX512-based vectorizations.

6.3.2 Standardized parallel algorithms

As second ingredient to our many-core parallel algorithms, we expect to have access to a parallel library of standardized (many-core parallel) algorithms similar to the C++ Standard Template Library (STL) algorithms library. We e.g. need `reduce`, `stable_sort`, `scan`, ... These algorithms are expected to be realized as one function call that is executed on data in global memory. The many-core parallel implementation of these algorithms is assumed to be extremely optimized and given e.g. by the vendor. On GPUs an implementation of STL-like algorithms is available via the `Thrust` library [BH11]. Alternatives include, but are not limited to `ArrayFire` [YAM+15] (supporting GPUs, CPUs and Xeon Phi) and `Boost.Compute` [Szu16] (supporting multi-core CPUs and GPUs). In addition, we assume to have appropriate BLAS libraries for a given many-core device.

6.4 Many-core parallel algorithmic patterns for $\mathcal{H}$ matrices

In the following, we introduce parallel patterns and algorithms for the many-core parallel construction of $\mathcal{H}$ matrices and the many-core parallel $\mathcal{H}$ matrix-vector product.

6.4.1 Spatial data structures and clustering based on Z-order curves

We use a Z order space filling curve [Mor66] to introduce a spatial data structure on top of the input point set $\mathcal{Y}$. This idea is based on [LGS+09]. We assign each point in $\mathcal{Y}$ a *Morton code*, which is an integer value. By ordering the elements of $\mathcal{Y}$ following their Morton codes, two consecutively ordered points get spatially close to each other, cf. Fig. 6.1 on the left-hand side. In the following, we will always assume that the input point set $\mathcal{Y}$ is stored following the Morton order. The implicit spatial structure introduced by the Morton ordering strongly simplifies the construction of the cluster tree: Whenever we have to split up a given cluster into two spatially distinct clusters in cardinality-based clustering, we only have to divide a given ordered point array into two parts, i.e. the first half of the elements builds the first cluster

Algorithm 11 Computation of Morton codes

```
procedure COMPUTE_MORTON_CODES<|𝒴|>(coords)
    for each thread t = 0, ..., |𝒴| − 1 in parallel do
        current_code ← 0
        for i = 1, ..., d do
            code_current_dim ← COMPUTE_FIXED_POINT_REPRESENTATION(coords[i][t])
            code_current_dim ← STRETCH_BITS(code_current_dim, i, d)
            current_code ← INTERLEAVE(code_current_dim,current_code, i)
        morton_codes[t] ← current_code
    return morton_codes
```

and the second half of the elements builds the second cluster, cf. Fig. 6.1 on the right-hand side. That is, spatial operations get reduced to array operations. To guarantee this ordering in the whole algorithm, we construct the Morton code and the reordering once and permute the vector $\boldsymbol{x}$ in the $\mathcal{H}$ matrix-vector product appropriately.

Note that the construction of point clusters based on Z-order curves might not result in clusters with the exact same numerical quality as in the case of the construction based on e.g. k-d trees. That is, a lowered quality of the clusters might have an impact of the required computational work of the method. However, as we will show, this cluster construction methodology leads to a very fast and very parallel procedure giving high performance to the method. A theoretical and empirical analysis of the impact of the use of space filling curves for clustering is considered future work.

On many-core hardware, we follow the lines of [LGS+09] and compute the Morton codes for a point set in a trivially parallel way.[3] Algorithm 11 summarizes the corresponding parallel kernel COMPUTE_MORTON_CODES. Per parallel thread / point coordinate, it iterates over the dimensions of the point coordinates, where it transforms the floating-point representation of the coordinate entry to a fixed-point representation. Next, the bits of the fixed-point representation are stretched. Finally, the stretched bits are interleaved dimension-wise such that the final Morton code is constructed. Sorting the points following their Morton codes is an operation of log-linear complexity for which we assume to have an STL-like operation, cf. Section 6.3.2.

From a data structure point of view, we collect the points $\mathcal{Y}$ in instances of a struct `point_set`. The struct contains a multi-dimensional array `coords` of point coordinates in Morton order, the dimension of the points and the number of points $|\mathcal{Y}|$. As described in Section 6.2, the $\mathcal{H}$ matrix method strongly relies on sub-blocks $A_{\phi,\mathcal{Y}\times\mathcal{Y}}|_{\tau\times\sigma}$ of matrix $A_{\phi,\mathcal{Y}\times\mathcal{Y}}$, which are constructed over index blocks $\tau\times\sigma\subset I\times I$. As we will see, clusters $\tau\subset I$ will always correspond to points that are (by Morton ordering) consecutively stored in `coords`. Therefore, we can define a cluster τ by index ranges $\{i_{l,\cdot}, i_{l,\cdot}+1, i_{l,\cdot}+2, \ldots, i_{u,\cdot}\}$ pointing to the storage location in `coords`. That is, each cluster τ is represented just by the lower and upper index bounds $i_{l,\cdot}$ and $i_{u,\cdot}$. Moreover, we collect the nodes $w\in V_{I\times I}$ of the block cluster tree $\mathcal{T}_{I\times I}$ in instances of structs `work_item`. In addition to the lower and upper index bounds for clusters τ and σ, this struct defines storage for bounding boxes of the clusters τ, σ and an admissibility

[3] We here assume that the reader has some knowledge about the construction of Morton codes. For details, see e.g. [BET99].

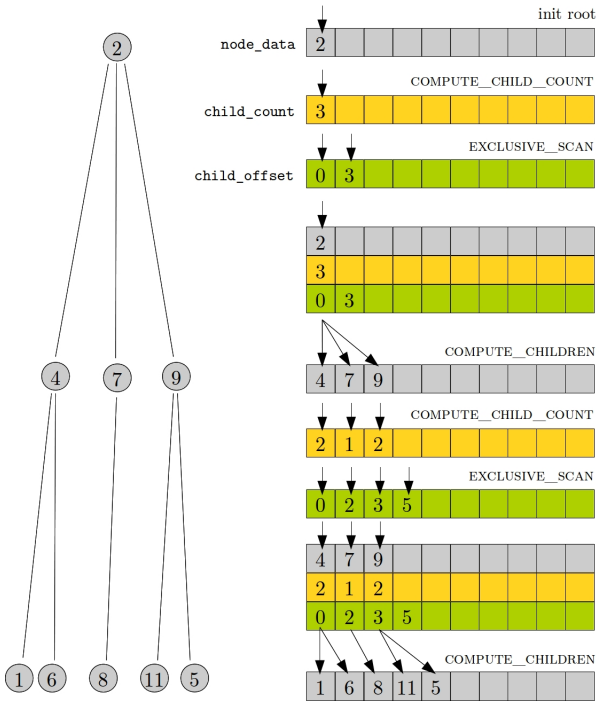


Figure 6.2: The many-core parallel block cluster tree traversal parallelizes over the nodes on a given level of the tree. The parallel algorithm is here exemplified for the strongly simplified situation of a tree with nodes containing numbers (instead of products of clusters).

flag.

6.4.2 Block cluster tree traversal

In the following, we introduce a fully parallel tree construction and traversal algorithm for the block cluster tree $\mathcal{T}_{I\times I}$. The algorithm is related to ideas in [LGS+09, MGG12] and on-the-fly builds and traverses a block cluster tree. An input tree $\mathcal{T}_{I\times I} = (V_{I\times I}, \gamma, \mu)$ is assumed to have height $h(\mathcal{T}_{I\times I})$, levels $l \in \{0, \ldots, h(\mathcal{T}_{I\times I})\}$ and block clusters or *nodes* $w \in V_{I\times I}$. The algorithm is designed such that we only store nodes $V_{I\times I}(l) := \{w \in V_{I\times I} \,|\, level(w) = l\}$ and $V_{I\times I}(l+1)$ for two consecutive levels l and $l+1$. All other data is created level-wise and discarded after a new level has been successfully created. The nodes $w \in V_{I\times I}(l)$ and $w' \in V_{I\times I}(l+1)$ are stored in global arrays `node_data_old` and `node_data`. In the construction of level $l+1$, we further need the number of children per node $w \in V_{I\times I}(l)$, i.e. $|\gamma(v')|$, stored in `child_count` and the offset of the data of the child nodes (`child_offset`). Figure 6.2 illustrates these arrays.

Our approach, as given in Algorithm 12, works as follows: Let us assume for now that the arrays per level can have arbitrary size and that we are on level $0 \leqslant l < h(\mathcal{T}_{I\times I})$ and the only available data is the node data $V_{I\times I}(l)$, stored in `node_data_old`. We first compute bounding box information for each node / block cluster in $V_{I\times I}(l)$. This is done using some lookup table

Algorithm 12 Many-core parallel block cluster tree traversal

```
procedure TRAVERSE(root)                ▷ Traverse a block cluster tree with given root data
    allocate node_data, node_data_old, child_count, child_offset
    node_data[0] ← root_data
    node_data_old ← node_data
    l ← 0
    |V_{I×I}(l)| ← 1
    while |V_{I×I}(l)| > 0 do           ▷ Handle block cluster tree levels as long a there are nodes
        (compute bounding box information bnd_info, see Section 6.7)
        COMPUTE_CHILD_COUNT<|V_{I×I}(l)|>(child_count, node_data_old, bnd_info)
                                        ▷ Kernel to compute the number of children per node
        EXCLUSIVE_SCAN(child_offset, child_count, 0, |V_{I×I}(l)|)
        |V_{I×I}(l+1)| ← child_offset[|V_{I×I}(l)|]          ▷ Set total number of children
        COMPUTE_CHILDREN<|V_{I×I}(l)|>(node_data, node_data_old, child_count, child_offset)
                                        ▷ Kernel to compute the content of the children
        node_data_old ← node_data
        l ← l + 1
```

mechanism and a technique called *batching* that we outline in greater detail in Section 6.4.3. Details on the bounding box calculation are given in Section 6.7. Then, we invoke a kernel COMPUTE_CHILD_COUNT with the number of threads equal to the number $|V_{I\times I}(l)|$ of nodes on that level. In the kernel, we independently evaluate for each node $w \in V_{I\times I}(l)$ (being an instance of a struct `work_item`, cf. Section 6.4.1) the admissibility condition (6.3) using the precomputed bounding boxes. Depending on the result, the number of children $|\gamma(w)| \in \{0, 4\}$ on the next level is written into `child_count` at the same offset in the array as the given node data. In a next step, we have to compute the offsets for the node data on the next level, i.e. `child_offset`. This can be done by an EXCLUSIVE_SCAN operation initialized to 0, cf. Fig. 6.2. The entries of `child_offset` then become `[0, child_count[0], child_count[0]+child_count[1],...]`. The output of the scan operation contains as additional number (at the end of the set of valid entries) the total number of children $|V_{I\times I}(l+1)|$.

The last step on level l is the creation of the node data $V_{I\times I}(l+1)$ on level $l+1$. This is again done using a kernel with the number of threads equal to $|V_{I\times I}(l)|$. Each thread then independently computes the new entries taking the storage location in `node_data` for level $l+1$ from `child_offset`. In particular a thread either creates new children by splitting up the index set by CBC clustering based on the Morton order or, if a node turns out to be a leaf node, puts the node as admissible or non-admissible leaf node to a *write-only parallel output queue* `work_queue` of `work_item` structs.

The parallel output queue is a global memory array of appropriate size. Additionally, we store a pointer to the head and the tail of the queue in global memory. We fill this queue in parallel, i.e. it grows during the block cluster tree traversal. Whenever a PUT operation is issued in a thread of a kernel, the head pointer is moved accordingly by an *atomic* operation while storing the old head in the same operation. The old head is used as output address to write the data in the queue. Figure 6.3 summarizes and exemplifies the approach. Removal of data or reading the head of the queue during the enqueueing process is not required. The actual

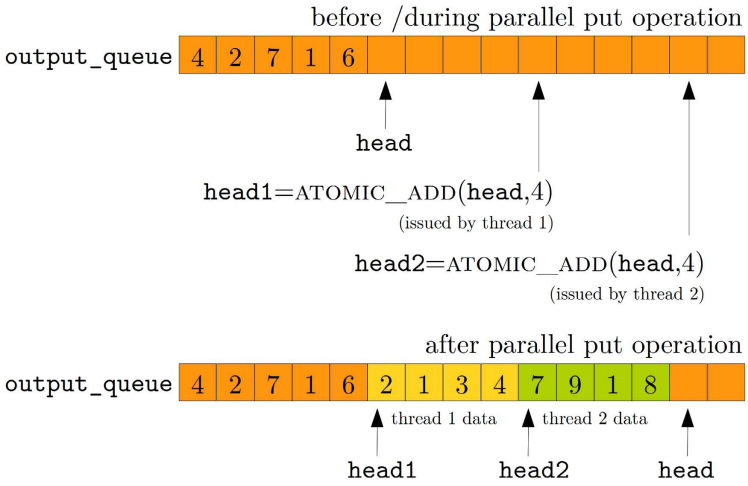


Figure 6.3: By the use of atomic operations it is easily possible to create a write-only parallel output queue. In the above example, two threads concurrently add four entries to the head of the queue.

execution of the tasks, stored in the output queue, is done later in the $\mathcal{H}$ matrix construction and is an operation on a fixed-size array.

This finishes the computation for one level. The whole process is iteratively proceeded over all levels $0 \leqslant l < h(\mathcal{T}_{I\times I})$. To start the tree traversal on level zero, i.e. the root of the tree, the `node_data` array is initialized with a single entry containing the set $I \times I$.

We next have to discuss how to deal with the array allocation in presence of changing level sizes. The first option would be to pre-allocate the arrays to a fixed size $\max_{l\in\{0,\dots,h(\mathcal{T}_{I\times I})\}} |V_{I\times I}(l)|$. This requires to know this number beforehand or to have a suitable upper bound for it. Very often, this is not the case. The second option is a *dynamic* allocation of the array size for the next level. This size can be predicted based on the information in the `child_count` array. In case a re-allocation of memory is a very expensive operation on a given target architecture, one could also apply hybrid approaches such as adapting the size of the arrays only if a given array (of large size) would be too small for the next level. In our implementation on GPU, a global re-allocation of the memory is a very efficient operation. This is why we have chosen to use the dynamic allocation approach.

Let us now discuss the utilization of the many-core processor for our algorithm. It becomes obvious that the number of used parallel threads on the first few levels is very low. Here, we make no full use of the many-core processor. This might become an issue, if many tree traversals on small trees are considered and if the tree traversal operation itself is the dominant operation in an application. However, both is not the case in our application: The block cluster tree is very large and, as we will see in Section 6.5, the tree traversal operation makes only a very small fraction of the overall $\mathcal{H}$ matrix setup / application process. Therefore, we consider our tree traversal method efficient enough for our needs. In case higher utilization of the many-core processor is needed, efficient solutions become very architecture-specific. In case of

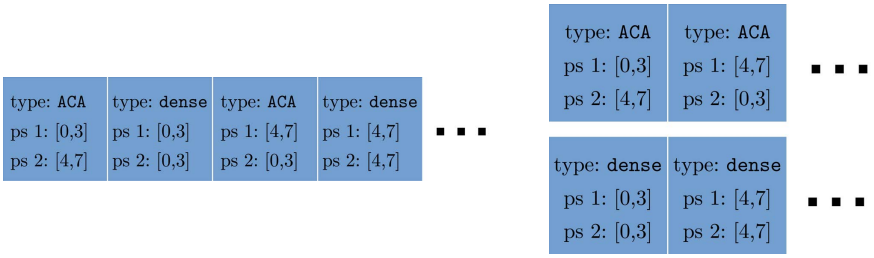


Figure 6.4: *Left:* The work queue generated during the block cluster tree traversal only contains meta information. No matrix element has been evaluated, yet. *Right:* The work queue is split up into admissible and non-admissible work elements (`ACA` vs. `dense`), before it is used as ordering for the batched linear algebra operations.

GPUs, there is work on tree traversal by work queues [GPM11], which, however, makes explicit use of knowledge on the hardware and somehow even breaks the programming model initially considered for (*NVIDIA*) GPUs.

6.4.3 Numerical linear algebra

During the block cluster tree traversal, an array `work_queue` of `work_item` structs is constructed (via the parallel output queue), cf. Fig. 6.4. It contains the matrix sub-block information of blocks that are either approximated by ACA or directly constructed as dense matrices, i.e. admissible or non-admissible leafs. Note that we did not evaluate a single matrix entry up to this point. So we only work on meta data. We initially decompose the `work_queue` into two according sub-arrays `aca_work_queue` and `dense_work_queue`, cf. Fig. 6.4. For the sub-matrices represented by the entries of these arrays, we either apply adaptive cross approximation or dense matrix-vector operations.

In classical (sequential) $\mathcal{H}$ matrix implementations, both, the factors U and V of the adaptive cross approximation *and* the dense matrix blocks are precomputed during an initialization phase and then stored in memory. This is due to the fact that often, e.g. in boundary element methods, the evaluation of a single matrix entry is already considered very expensive, a storage operation in memory is relatively cheap and large amounts of (CPU) memory are available. Using many-core processors, this balance is somewhat different. Here, evaluating matrix elements is often much faster. However storing data in global memory, i.e. not keeping it in the local memory of the kernel, is rather expensive. Moreover, the memory of many-core processors is often very limited. Therefore, we adapt the classical strategy to the abilities of many-core processors in the following way: We normally always re-compute all low-rank approximations and re-assemble dense matrices during each application of the fast matrix-vector product. Thereby we do not run into the very strong memory limitations of many-core processors. However, we also add the option to pre-compute the construction of the factors U and V in the adaptive cross approximation once, while using these factors during many matrix-vector products. Note however that this is very memory-consuming. A pre-computation of the dense sub-blocks is never done.

In the following, we first introduce the general concept of *batching* many small similar non-

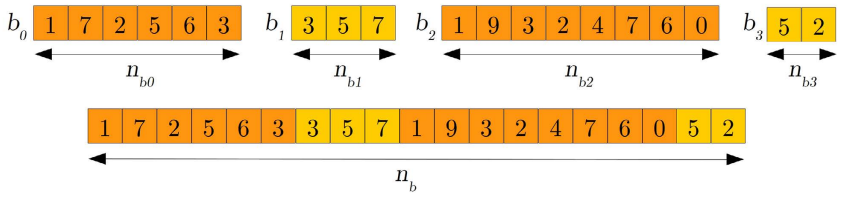


Figure 6.5: By batching individual subproblems into one big array, it becomes possible to utilize a many-core processor much better.

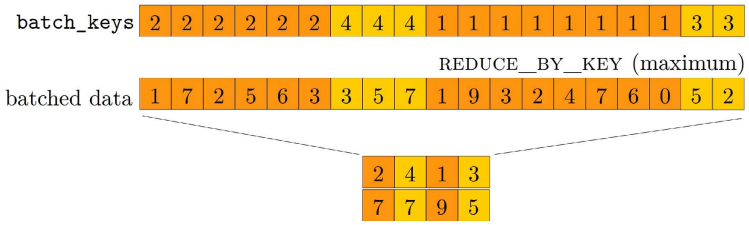


Figure 6.6: Reduction operations (here maximum computations) for several sub-problems can be handled in parallel by a REDUCE_BY_KEY operation, where identical consecutive entries in `batch_keys` mark the individual batches.

equally sized compute tasks. This concept is then heavily used in the computation and application of the adaptive cross approximation and the dense matrix-vector products.

Batching many similar non-equally sized compute tasks

In the numerical linear algebra operations showing up in $\mathcal{H}$ matrix calculations, there is often an identical computing task which shall be applied to m different, non-equally sized arrays $b_0, b_1, \ldots, b_{m-1}$ of sizes $n_{b_0}, n_{b_1}, \ldots, n_{b_{m-1}}$. Figure 6.5 gives an example of such arrays. The easiest way to consider a parallelization on many-core hardware would be to loop over all arrays b_i and to perform the necessary many-core parallel operations independently to each array. This is efficient as long as the many-core processor is sufficiently utilized. However, we here consider arrays of changing and, usually, small size. In this case, a major part of the many-core processor is not used. Therefore we propose to use the technique of *batching* of the necessary computations, cf. [CKL, AHTD17], in order to use the full processor while speeding up the calculation.

The first step in batching is to put all sub-arrays or *batches* b_i consecutively in a *batched array* of size $n_b := \sum_i n_{b_i}$, cf. Fig. 6.5. We next have to distinguish between *transformation* operations and *reduction* operations on that batched array. A transformation on each batch applies changes individually to each entry of each batch, i.e. there is no interaction between the data entries. Applying a transformation to each batch is therefore equivalent to applying the same transform to the full batched array. Therefore, in case of transformations, we apply one operation to the full batched array.

In contrast, reduction operations (such as sum, minimum, maximum, norm, etc.) require

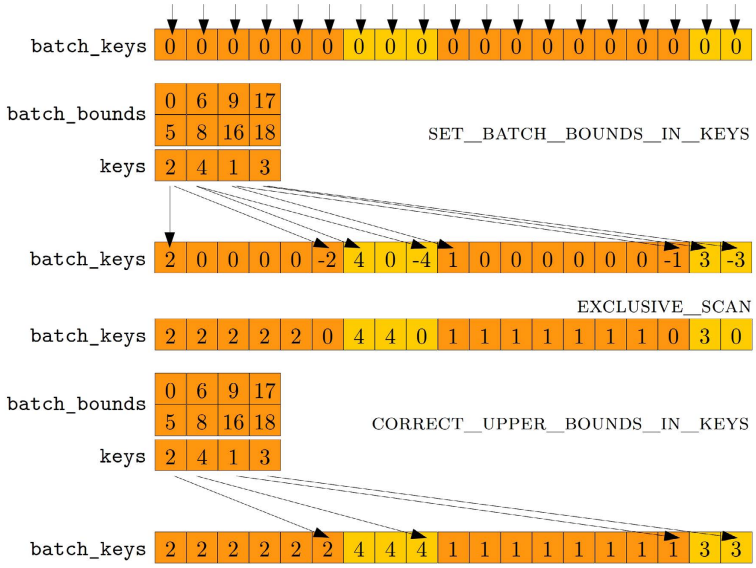


Figure 6.7: The construction of an array of keys for batching involves marking boundaries of the batches and an EXCLUSIVE_SCAN operation.

Algorithm 13 Many-core parallel key generation for batching

procedure CREATE_KEYS(`batch_bounds`, `keys`, n_b, m)
  INIT<n_b>(`batch_keys`, 0)
  SET_BATCH_BOUNDS_IN_KEYS<m>(`batch_keys`, `batch_bounds`, `keys`)
  EXCLUSIVE_SCAN(`batch_keys`, `batch_keys`, 0, n_b)
  CORRECT_UPPER_BOUNDS_IN_KEYS<m>(`batch_keys`, `batch_bounds`, `keys`)
  return `batch_keys`

the interaction of all entries within a batch. Therefore, we need a different strategy. The STL-type algorithm REDUCE_BY_KEY is applied to the full batched array and computes, in parallel, batch-wise reductions. The action of the method is shown in Fig. 6.6 for a maximum reduction operation. We introduce a `batch_keys` array of integer values. A consecutive series of identical numbers in the `batch_keys` array marks one batch. The method REDUCE_BY_KEY then applies the reduction operation per batch and builds up a small array of size m containing the reduction results and the keys reduced to a single number.

To compute the `batch_keys` array, we need an additional parallel algorithm, cf. Algorithm 13. It takes an array of boundaries (`batch_bounds`) of each batch b_i and an array (`keys`) of keys k_{b_i} per batch as input. The procedure to create keys for batching is exemplified in Fig. 6.7. We initialize (by a kernel of n_b threads) the `batch_keys` array to zeros. Then, the kernel SET_BATCH_BOUNDS_IN_KEYS of m threads is invoked, where each thread independently writes the key k_{b_i} and the negative key $-k_{b_i}$ to the lower and upper bound of each batch in the batched array, cf. Fig. 6.7. Then an EXCLUSIVE_SCAN operation (adding elements) is executed on the full batched array. This sets the correct keys almost everywhere, except for the upper

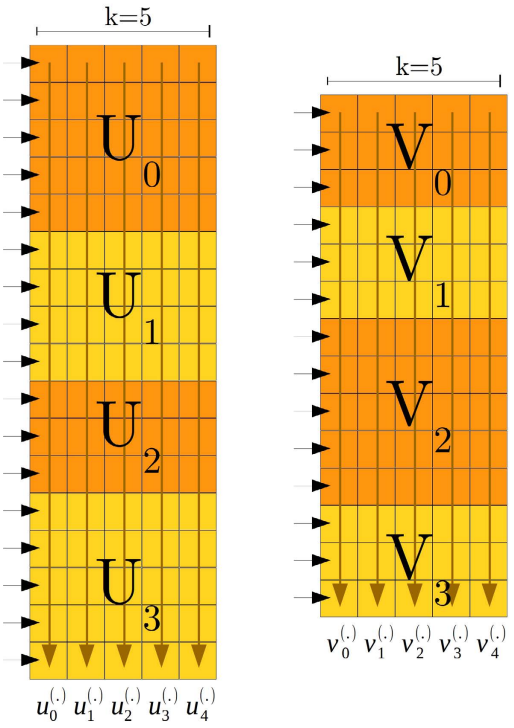


Figure 6.8: In the batched version of the adaptive cross approximation, the columns of the low-rank factors U_l and V_l are stored consecutively in memory. The brown arrows indicate the order in memory while the black arrows indicate threads in the parallelization.

boundary of each batch. Therefore a second kernel of m threads is invoked to correct the upper bounds of each batch b_i to k_{b_i}.

In some cases, the size n_b of the batched array is too large to be kept in the memory of the many-core processor. Such cases can be handled by appropriately partitioning the batches b_i into subsets of batches, which are then handled as before.

A crucial property of the approach presented here is its independence of the size and the number of batches. This is a strong advantage over strategies that directly rely on the use of the different parallelization hierarchies (*thread blocks*, *grids* on GPUs and *vectorization*, *shared-memory parallelism*, etc. on Xeon Phi).

Batched adaptive cross approximation

We apply batching to compute and apply adaptive cross approximations for all ACA elements in the `aca_work_queue`. The storage pattern is to consecutively store elements $\boldsymbol{u}_l^{(0)}, \boldsymbol{u}_l^{(1)}, \ldots \boldsymbol{u}_l^{(m-1)}$ in memory, where a single ACA sub-matrix $U^{(i)}$ is given as $U^{(i)} = (\boldsymbol{u}_1^{(i)} \boldsymbol{u}_2^{(i)} \ldots \boldsymbol{u}_k^{(i)})$. The top index is the batch number and l is the index of the rank-one information. The blocks of batched rank-one information is then stored consecutively for $l = 0, \ldots, k-1$, where k is the maximum

number of ranks that is initially given as user argument. Figure 6.8 shows this storage principle.

In the batched ACA computation, we first set up several meta data arrays describing mainly mappings between the batched ACA data, indices of the input point set and the work items in the `aca_work_queue`. These mappings are used to have constant-time access in kernels being parallelized over the points, over the `aca_work_queue` entries or over the batched ACA data. We can compute these maps similar to the approach presented e.g. in Algorithm 13. Then, we execute the classical ACA algorithm in a batched version. That is, simple transformations can be applied directly to the full batched array while batch-wise reductions are handled as described before.

Note that the ACA algorithm has an data-dependent iterative behavior, e.g. in case of pivoting. That is, the algorithm might need different numbers of iterations for different batches. We cope with this by introducing a voting mechanism that stops the iterative process when work is finished on all batches. A drawback of this approach is that the runtime for the batched version is bound from above by the *slowest* batch. However, from our practical tests, this has never been a performance issue.

Depending on the choice of pre-computing or directly applying the low-rank factors U and V, we either keep these factors in global memory for later use or we directly apply them using BLAS library calls for dense matrix-vector products.

If we choose to recompute the ACA during each matrix-vector product, we further have the opportunity to split up the whole batched ACA computation to several smaller batched ACA operations. This allows to handle much larger matrices, which would otherwise not fit into GPU memory. To make this possible, we have to choose the number m of matrix batches per batched matrix. We have designed a heuristics that fills up a batched matrix with matrices of size $n_{b_i} \times k$ as long as $\sum_i n_{b_i}$ is smaller than a threshold bs_{ACA}, i.e. the batching size for ACA. As we will see in Section 6.5.6, the choice of this batching size parameter is important for the performance of the code.

Batched dense sub-matrix application

The application of the dense sub-matrix matrix-vector products is also done in a batched, parallel way. Analogously to the batched ACA computation, we first assemble, entirely in parallel, a larger number of dense sub-blocks using an appropriate compute kernel. The storage principle is similar to the one presented in the previous paragraph, i.e. we stack the dense matrices of size $n_{b_i} \times n'_{b_i}$ on top of each other. To get a simpler representation in memory, we pad all batched sub-blocks by zero columns such that they have all the same column count $\max_i n'_{b_i}$. Afterwards, we use a batched version of *BLAS* for the dense matrix-vector products.

As in the case of batched ACA computation, we have designed a heuristics to create batches of fixed maximum size. In case of the batched dense matrix-vector products, we choose to keep the total batch storage size smaller than a threshold bs_{dense},

$$\max_i n'_{b_i} \cdot \sum_i n_{b_i} \leqslant bs_{dense} \,.$$

6.5 Results

In this section, we evaluate the performance of the above described many-core parallel algorithms in the concrete GPU implementation `hmglib` [Zas18] developed by the author. The library is available via GitHub and is licensed under LGPL License Version 3.0. This implementation only covers the $\mathcal{H}$ matrix construction or setup and $\mathcal{H}$ matrix vector product for a matrix $A_{\phi,\mathcal{Y}_1\times\mathcal{Y}_2}$ for a given kernel function ϕ and sets $\mathcal{Y}_1$ and $\mathcal{Y}_2$. It is not intended to be feature-complete, i.e. providing the full $\mathcal{H}$ matrix algebra. Instead, it is a test bed for the above discussed many-core parallel algorithms. Nevertheless, it is possible to solve linear systems of type (6.1) by using the iterative dense linear solvers library `MPLA` [Zas17] developed by the author (Open Source, available on GitHub), which has an interface to `hmglib`. However, the objective of this benchmark chapter is to stick to the discussion of the construction and the $\mathcal{H}$ matrix-vector products, avoiding to confuse the reader with solver details and with two different library implementations.

In the following, we start our discussion by giving brief details on the library `hmglib` with the targeted hardware and applied external many-core parallel libraries. Afterwards, we introduce a model problem and show empirically that the implemented approximate matrix-vector product converges dimension-dependent almost exponentially in the rank k used in the adaptive cross approximation for the given model problem. Since the main goal is, to get a code of optimal complexity, we check the runtime complexity of `hmglib` by numerical experiments. Thereafter, we give details about the performance improvements made by batching. As we will see, these performance improvements have the highest impact on our final results. We finish this section by comparing the runtimes of `hmglib` against a reference CPU implementation. Note here, that we will compare a multi-core multi-purpose state-of-the-art, Open Source library for hierarchical matrices (`H2Lib` [BÏ7]) with a very specific, parallel many-core implementation on roughly equally priced hardware. Comparisons like this can never be completely fair. Therefore, the results of this study are only treated as a rough hint towards the actual performance improvement by using `hmglib`.

6.5.1 GPU implementation `hmglib`

The library `hmglib` [Zas18] is implemented for graphics processing units of *NVIDIA Corporation*. Our notion of a compute kernel from Section 6.3.1 can be easily mapped to the compute kernels in the C language extension *CUDA* for programming *NVIDIA* GPUs. Note however, that an implementation in *OpenCL* (for *NVIDIA* and *AMD* GPUs) or *OpenMP* with extensions for *Intel Xeon Phi* devices should be equally possible. Within our hand-implemented *CUDA* compute kernels, we always use a so-called *block size* of 512, i.e. 512 threads are bundled in a block with common shared memory (which we actually do not explicitly use). `hmglib` uses the *CUDA Toolkit 8.0*. It is compiled with optimization parameter `-O3`. The CPU code compiler will be discussed later.

Within our many-core parallel algorithms in Section 6.4, we launch, beside of compute kernels, library calls for general many-core parallel STL-type algorithms. In `hmglib`, the library `Thrust`, which is delivered as part of the *CUDA Toolkit*, provides these STL-type algorithms. `Thrust` contains all the necessary parallel algorithms and delivers decent performance for GPUs. Moreover, we use BLAS-type operations of the library `CUBLAS`, which is also delivered as part of

the *CUDA Toolkit*. In case of the batched application of dense matrix-vector products, we apply the state-of-the-art GPU *Lapack* library `Magma 2.2.0`. There, we specifically use the batched multiplication
`magmablas_dgemv_vbatched`.

`hmglib` allows to select, whether batching is applied in the matrix-vector product, or not. Moreover, it is possible to switch on the pre-computation of the low-rank factors in the adaptive cross approximation. This requires a lot of GPU memory. However, $\mathcal{H}$ matrix-vector products can be applied faster if the low-rank factors do not have to be recomputed for each multiplication. Remember that in CPU-based $\mathcal{H}$ matrix implementations, the dense sub-blocks of the approximated matrix are often pre-computed, too. This is not done here, due to limited GPU memory and very fast matrix assembly on GPU. All calculations are done in double precision.

6.5.2 Hardware setup and time measurements

While a major part of the development work has been carried out on the cluster Titan at Oak Ridge National Lab, the benchmarking was done on the *PSG Cluster* of *NVIDIA Corporation*. On the latter one, IBM S822LC compute nodes with *IBM POWER8* architecture were used. They are each equipped with two 10-core *IBM POWER8* processors at 2.86 GHz leading to a total of 160 logical cores on a Linux system, 512 GB RAM and four NVIDIA Tesla P100 SXM2. Only one out of these four GPUs was used. Our CPU performance comparison is done on the same platform. Additionally, we give timings for an Intel compute server equipped with two 20-core Intel Xeon E5-2698 v4 CPUs with 2.20 GHz and 768 GB RAM. Hyper-threading is switched on on the Intel machine leading to 80 logical cores. Note that the price for the two Intel CPUs seems[4] to be in the same range as the price of a single Tesla P100 SXM2.

Whenever we use GPU-based calculations, we use *CUDA Events* to get very accurate time measurements. The time required by potentially necessary data transfers between GPU and CPU is always included. However, we assume the initial data, i.e. the point set $\mathcal{Y}$ to reside in GPU memory. In case of CPU-based $\mathcal{H}$ matrix benchmarks, we use the *gettimeofday* command to do the measurements. All measurements (GPU and CPU) are averaged results over five trials of a $\mathcal{H}$ matrix construction or a $\mathcal{H}$ matrix-vector product with different random vectors $\boldsymbol{x}$.

6.5.3 Model problem

All benchmarks consider matrix-vector products of the form

$$A_{\phi,\mathcal{Y}\times\mathcal{Y}}\boldsymbol{x}$$

with

$$A_{\phi,\mathcal{Y}\times\mathcal{Y}} = \begin{pmatrix} \phi(\boldsymbol{y}_1,\boldsymbol{y}_1) & \cdots & \phi(\boldsymbol{y}_1,\boldsymbol{y}_N) \\ \vdots & \ddots & \vdots \\ \phi(\boldsymbol{y}_N,\boldsymbol{y}_1) & \cdots & \phi(\boldsymbol{y}_N,\boldsymbol{y}_N) \end{pmatrix}, \quad \boldsymbol{x} \in \mathbb{R}^N .$$

[4] An exact pricing for the Tesla P100 SXM2 was, at time of writing this article, hard to find, since it is no discrete graphics card. The discrete Tesla P100 version with 16 GB has an identical pricing as the two Intel CPUs.

where $\mathcal{Y} := \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\} \subset \Omega$ is a set of N points in a space $\Omega \subset \mathbb{R}^d$ and $\phi : \Omega \times \Omega \to \mathbb{R}$ is a bivariate kernel function operating on that domain. We specifically choose $\Omega = [0,1]^d$ with $d = 2, 3$. Moreover, the point set is a Halton sequence, i.e. a quasi Monte-Carlo sequence, of length N in d dimensions. This choice corresponds to the typical setup in kernel-based approximation on the unit square / cube. We test the implementation with different (unscaled) kernel functions, namely the Gaussian kernel

$$\phi_G(\boldsymbol{y}, \boldsymbol{y}') = e^{-\|\boldsymbol{y}-\boldsymbol{y}'\|^2}$$

and a Matérn kernel [Fas07, Section 4.4]

$$\phi_M(\boldsymbol{y}, \boldsymbol{y}') = \frac{K_{\beta-\frac{d}{2}}(\|\boldsymbol{y}-\boldsymbol{y}'\|)\|\boldsymbol{y}-\boldsymbol{y}'\|^{\beta-\frac{d}{2}}}{2^{\beta-1}\Gamma(\beta)},$$

where K_ν is the modified Bessel function of second kind of order ν and Γ is the gamma function. We choose $\beta - \frac{d}{2} = 1$. The resulting matrix $A_{\phi_M,\mathcal{Y}\times\mathcal{Y}}$ shows up in first-order convergent function interpolation schemes in kernel-based interpolation [Fas07, Theorem 14.5, Example 15.4] for appropriately smooth functions. The norm $\|\cdot\|$ is the usual Euclidean norm of appropriate dimensionality.

This model problem represents the application fields of mesh-free kernel-based approximation, (non-regularized) kernel ridge regression and, in some cases, Gaussian process regression.

6.5.4 Convergence of the matrix-vector product approximation

We start our experiments by checking the convergence of our $\mathcal{H}$ matrix implementation for growing ACA rank k for all discussed kernel functions in two and three dimensions and problem size $N = 32768$. Furthermore, we choose $C_{leaf} = 256$ and $\eta = 1.5$. All other parameters are not relevant for this convergence study. As for the performance measurements, we perform five runs and average over each result. The error in each run is the relative error

$$e_{rel} = \frac{\|\mathcal{H}(A_{\phi,\mathcal{Y}\times\mathcal{Y}})\boldsymbol{x}_{rand} - A_{\phi,\mathcal{Y}\times\mathcal{Y}}\boldsymbol{x}_{rand}\|_2}{\|A_{\phi,\mathcal{Y}\times\mathcal{Y}}\boldsymbol{x}_{rand}\|_2}$$

for a random input vector $\boldsymbol{x}_{rand}$. $\mathcal{H}(A_{\phi,\mathcal{Y}\times\mathcal{Y}})$ is the $\mathcal{H}$ matrix approximation of the full system matrix $A_{\phi,\mathcal{Y}\times\mathcal{Y}}$. Note that we are strongly limited in the problem size N since we do all computations on GPU and therefore have to do the full matrix vector product $A_{\phi,\mathcal{Y}\times\mathcal{Y}}\boldsymbol{x}_{rand}$ in GPU memory.

Fig. 6.9 shows on the left-hand side the convergence results for $d = 2$ and the two different kernels from the model problem. For the Gaussian kernel, it is known that the error decays as $\exp\left(-\alpha\sqrt[d]{k}\right)$. Our implementation delivers this dimension-dependent almost exponential convergence in the number of ranks k used in the adaptive cross approximation. The same test is repeated for dimension $d = 3$ with similar results. Since the results for Gaussian and Matérn kernel are almost identical, we will restrict ourselves to performance studies for the Gaussian kernel.[5]

[5] Note that we stick to low dimensions d since it is well-known that these can be handled well by $\mathcal{H}$ matrices.

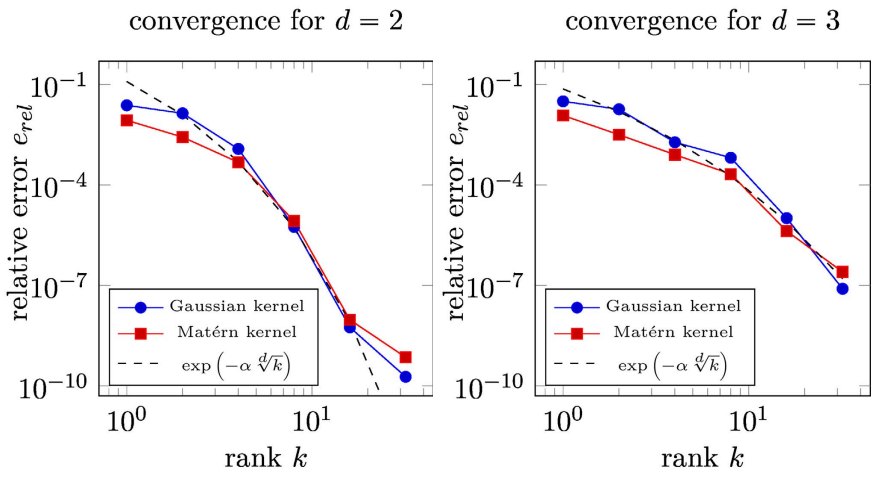


Figure 6.9: For fixed problem size $N = 32768$ and growing number of ranks in the adaptive cross approximation, the $\mathcal{H}$ matrix-vector product converges to the full matrix-vector product with the expected dimension-dependent almost exponential convergence for dimensions $d = 2$ (*left*) and $d = 3$ (*right*).

6.5.5 Runtime complexity and performance of the GPU-parallel code

The crucial objective of an implementation of the hierarchical matrix method is to achieve the optimal runtime complexity of $O(N \log N)$ for the matrix-vector product at fixed rank k. However, very often, high (pre-asymptotic) runtime performance on many-core hardware is only achieved by sticking to algorithmic simplifications of worse complexity but higher performance. The following empirical study shall show that the $\mathcal{H}$ matrix implementation in `hmglib`, which is based on our many-core parallel $\mathcal{H}$ matrix algorithms from Section 6.4, actually achieves the required $O(N \log N)$ runtime complexity. To study this, we choose $\eta = 1.5$, $C_{leaf} = 2048$, $k = 16$, $bs_{dense} = 2^{27}$ and $bs_{ACA} = 2^{25}$, use batching and carry out performance measurements for growing problem size N.

We first discuss the runtime complexity of the setup of the spatial data structure. While computing the Morton codes for all points $\boldsymbol{y}_i$ is of complexity $O(N)$, sorting the points following the Z order curve is a $O(N \log N)$ operation. This is reflected by our empirical study shown on the left-hand side of Fig. 6.10. For $d = 2$ and $d = 3$ we observe a runtime complexity of (below) $O(N \log N)$ after some pre-asymptotic range. The same behavior is observed for the construction and the traversal of the block cluster tree. Runtime results for this case are given on the right-hand side of Fig. 6.10. Note again that it is non-trivial to get the optimal complexity for such a complex many-core parallel code. Figure 6.10 further outlines that the spatial data structure setup and the tree traversal is actually very fast. Even for $N = 2^{26}$, i.e. an approximation of a dense matrix of roughly 67×67 million entries, we only need about 0.4 seconds for the spatial data structure and about 3 seconds for the tree traversal (for $d = 3$).

The second part of this runtime complexity study covers the application of the fast matrix-

The application of $\mathcal{H}$ matrices in high dimensions is ongoing research.

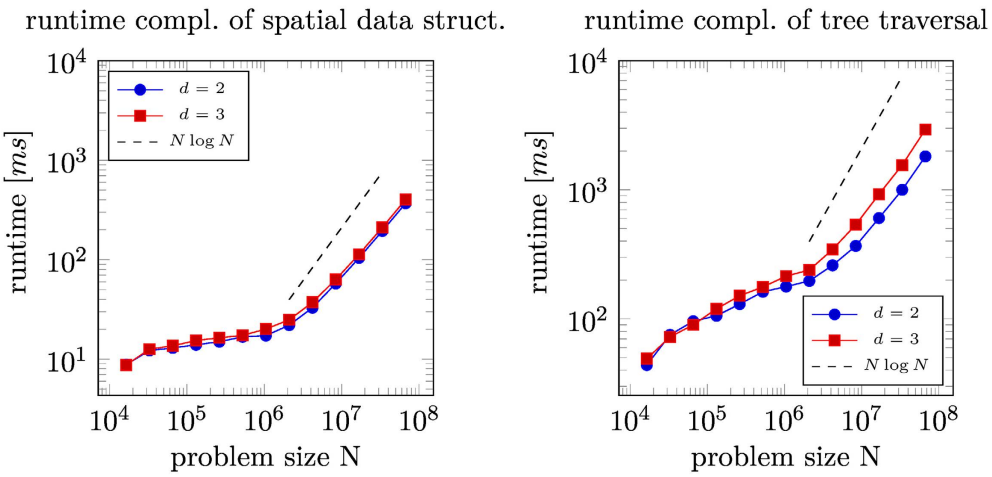


Figure 6.10: Even for the largest problem size of $2^{26} \approx 67$ million unknowns, the time for the spatial data structure setup is below 0.5 seconds (*left*). The tree construction and traversal requires less than 3 seconds for 2^{26} unknowns, while being faster than the required runtime complexity of $N \log N$ (*right*).

vector product. Figure 6.11 shows the measurements of the runtime for growing problem size N and different dimensionality d. Within each performance plot, we further distinguish between measurements that were done using a matrix-vector product with precomputed ACA factors and with on-the-fly computation of the ACA factors. Pre-computing the ACA factors results in a performance improvement, which will be discussed in more detail in Section 6.5.7. In the plot, the impact is not clearly visible due to the logarithmic scaling of the vertical axis. We cannot show runtime results with pre-computing for problem sizes beyond $N = 2^{19}$ or $N = 2^{20}$ due to the limited GPU memory. For $d = 3$ in Figure 6.11, we further include (gray) semi-logarithmic plots indicating the runtime per degree of freedom and the respective $N \log N$ scaling. This shall support readers that are more familiar to this metric.

In all cases, we observe a runtime complexity of $O(N \log N)$. Moreover, even for a problem size of $N = 2^{25}$, i.e. an approximated matrix-vector product for a dense matrix of 33×33 million entries, we see a runtime of only 6 minutes for a matrix-vector product on points in two dimensions. To be concise, we skipped here a performance study with respect to the rank k. Here, we expect the typical quadratic complexity in k stemming from (re-)computing ACA. We also skipped a further study for parameter η that basically balances the amount of work in the dense blocks against the work in the low-rank blocks.

6.5.6 Performance analysis of batching

Beforehand, we discussed the performance results of our implementation using batching in all linear algebra operations, as discussed in Section 6.4.3. However, it is important to know that batching is one of the crucial ingredients of this code allowing for high performance of the overall method. To show the actual impact of batching, we will analyse the performance with

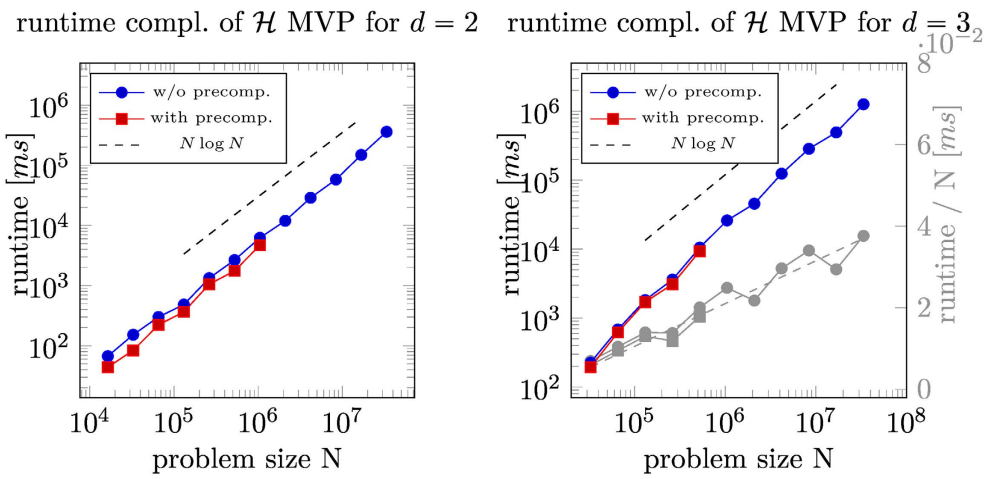


Figure 6.11: The $\mathcal{H}$ matrix-vector product shows the optimal algorithmic complexity of $O(N \log N)$. The operation is slightly more expensive if used on points in three dimensions (*right*) in contrast to points in two dimensions (*left*). By pre-computing ACA factors, performance is slightly improved. (Runtime per problem size is given in gray.)

and without batching in the linear algebra operations. Nevertheless, before we come to this point, we want to address the topic of parameter choice of the leaf size C_{leaf} and the batching sizes bs_{dense} and bs_{ACA} in connection with the batched operations.

Leaf size influence

The leaf size C_{leaf}, which has been introduced in Section 6.2.1, has a considerable influence on the amount of leafs / blocks that are generated in the block cluster tree. Clearly, the larger the leaf size the less blocks are generated, cf. Figure 6.12 on the left-hand size. Due to the quadratic complexity of the construction and application of dense blocks, the usual rule of thumb on CPUs is to keep the block sizes rather small. Here, a common choice for C_{leaf} seems to be $2k$ or $4k$, i.e. 32 and 64 for $k = 16$.

To understand the impact of the leaf size on *GPU-based* calculations with the proposed implementation, we perform a benchmark of the batched versions of the construction and application of the dense and ACA blocks in the $\mathcal{H}$ matrix-vector product with respect to different leaf sizes. We choose $N = 2^{20}$, $k = 16$, $\eta = 1.5$, $d = 2$, $bs_{dense} = 2^{25}$ and $bs_{ACA} = 2^{23}$. Figure 6.12 shows the results for this study with leaf sizes $C_{leaf} \in \{32, 64, 128, \ldots, 4096\}$. On the left-hand side, the number of blocks is reported. As expected, it drops for larger leaf size. The right-hand side of Figure 6.12 shows the runtime to construct and apply these blocks. The first observation is that the amount of runtime spent in the dense blocks grows for larger leaf size. This can be explained by the increased size of each dense block and the quadratic runtime complexity in the block size. In contrast, the runtime spent in the low-rank blocks drops for growing leaf size. There are two reasons for this behavior. First, the overall amount

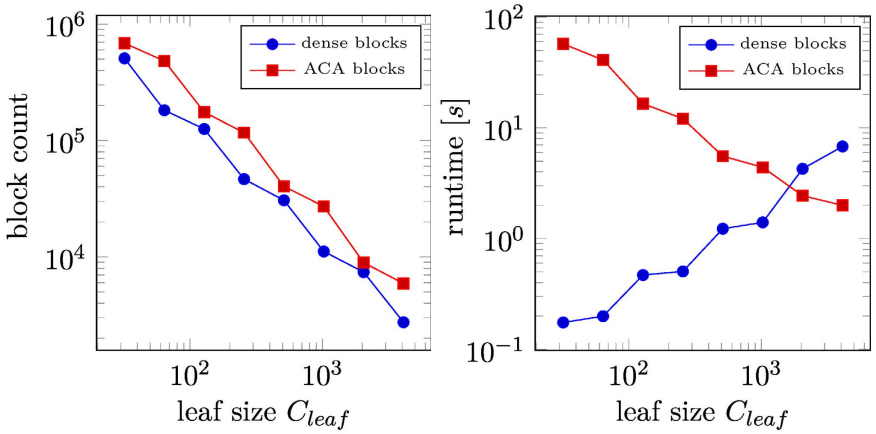


Figure 6.12: An increase in the leaf size C_{leaf} reduces the number of leafs in the block cluster tree *(left)*. At the same time, each leaf block gets larger. This leads to an increase in the total runtime spent in dense blocks, while the batched ACA block calculation becomes cheaper *(right)*.

of computational work is reduced, since larger dense blocks "cover" parts of the system matrix. Second, batching for ACA blocks is not as effective as for the dense block. That is, many small ACA blocks require more runtime than few larger ACA blocks of similar total size.

The runtime results in Figure 6.12 indicate that the usual choice of $C_{leaf} = 2k$ or $C_{leaf} = 4k$ leads to prohibitively large runtimes. This is due to the considerably faster computation times of the dense blocks on a GPU. The best total runtime is achieved for C_{leaf} in the range of 1024 and 2048. While not being shown here, similar runtime studies were done for other problem sizes N. These substantiate that the choice of C_{leaf} in the range of 1024 and 2048 is indeed rather optimal. In the following studies, we will stick to these two choices.

Batching size influence

In Section 6.4.3, we introduced the parameters bs_{dense} and bs_{ACA} as batching sizes for the batching of the dense matrix-vector products and the batching of the adaptive cross approximation. Remember that the *batching size* is not exactly the amount of ory that is allowed to be used in the batched dense / ACA operation. In general, these parameters balance the memory consumption against the performance improvement. To understand this relationship further, we benchmark the runtime of the batched dense matrix-vector products and the batched ACA in the $\mathcal{H}$ matrix-vector product for different batching sizes. It is done for $N = 2^{20}$, $k = 16$, $\eta = 1.5$ and $d = 2$. We consider results for $C_{leaf} = 1024$ and $C_{leaf} = 2048$, as discussed before.

Figure 6.13 collects the results for the parameter studies in the batching size for the batched dense matrix-vector products on the left-hand side and for the batched ACA computation on the right-hand side. Still, the choice of the leaf size C_{leaf} has a considerable influence on the performance balance between dense matrix-vector products and ACA. That is, larger leaf sizes lead to larger runtimes in the dense matrix-vector products. However the ACA runtime is

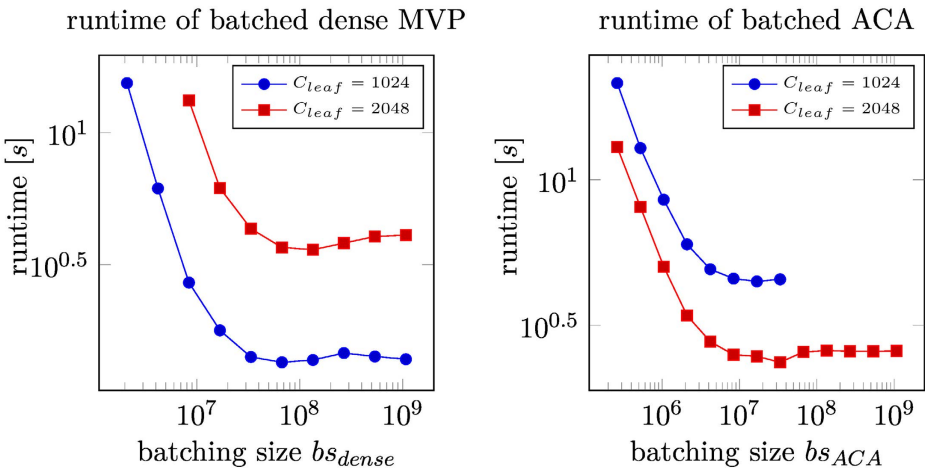


Figure 6.13: The performance of batching strongly depends on the size of the batched array or matrices that are used. This is clearly visible for the batching of dense matrix-vector products (*left*) and adaptive cross approximation (*right*). The optimal batching size is only slightly influenced by the choice of the parameter C_{leaf}.

reduced. The opposite holds for smaller leaf sizes. Note further that we could not perform runtime measurements for the smaller leaf size of $C_{leaf} = 1024$ and bs_{ACA} beyond 2^{25} in batched ACA, since, in this case, the amount of memory required in the storage of the ACA factors becomes larger than the available memory on the GPU. In contrast, this was possible for $C_{leaf} = 2048$, since we do not store the dense blocks in memory and less memory is spent in the low-rank factors.

The general tendency in the results in Fig. 6.13 is that increasing the batching size increases the performance up to an optimum. Beyond this optimum, the performance of the batching gets slightly worse. This performance improvement up to an optimum is due to the improvement of the occupancy of the GPU. That is, the GPU gets more work to do. Thereby, it can hide latencies etc. *behind* parallel work. The slight performance degradation beyond the optimum for larger batching sizes is maybe due to a slight over-subscription of the GPU: The maximum throughput limit is hit, however, due to more batches per batched operation, the data structure overhead becomes visible. Note, however, that this latter reasoning is speculative.

Overall, choosing an appropriate batching size is rather simple. The rule of thumb is to take it as large as possible (in terms of memory consumption) and to accept the slight performance reduction for a too large batch size.

Performance improvement by batching

We next discuss the performance improvement for batched dense matrix-vector products and for batched ACA over their respective non-batched versions. We use parameters $N = 2^{20}$, $k = 16$, $\eta = 1.5$, $d = 2$, $C_{leaf} = 2048$, $bs_{dense} = 2^{27}$ and $bs_{ACA} = 2^{25}$. Figure 6.14 summarizes the results of this study with results for the batching of dense matrix-vector products on the left-hand side and results for the batched adaptive cross approximation on the right-hand side.

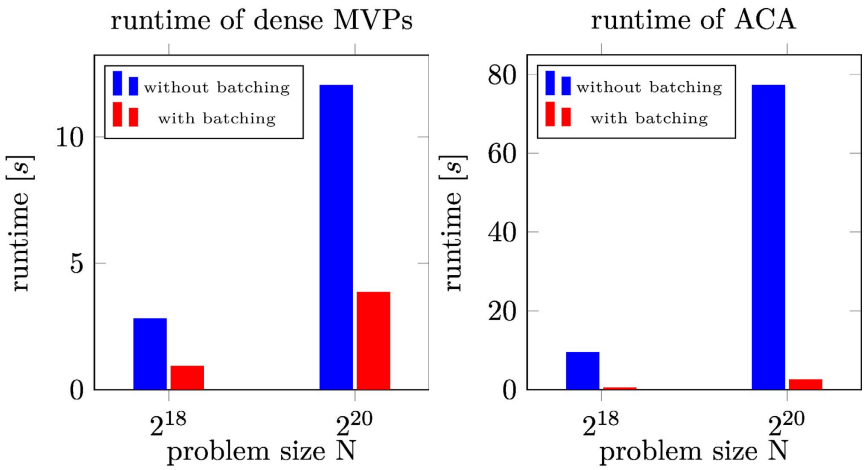


Figure 6.14: We observe a significant performance improvement by roughly a factor of 32, when using batching in the ACA computation (*right*). Batching dense matrix-vector products still improves performance by roughly a factor of three (*left*).

For a problem size of $N = 2^{20}$, the batched application of the dense matrix-vector products is by more than a factor of 3 faster. We do not gain more, since, for $C_{leaf} = 2048$, we have a lot of large dense matrix sub-blocks which very soon fully occupy the GPU.

In contrast, the performance improvement for the adaptive cross approximation is about a factor of 32 for $N = 2^{20}$. This strong impact is due to the small amount of work that is done for each individual ACA computation and is a significant contribution of this work.

To summarize, an efficient $\mathcal{H}$ matrix-vector product would not be possible without ACA batching. However, it also pays off to do batching for the dense matrix-vector products.

6.5.7 Performance comparison against `H2Lib`

In the following, we aim at relating the performance of `hmglib` to the CPU $\mathcal{H}$ and $\mathcal{H}^2$ matrix library `H2Lib` [BÏ7] in the, at time of writing this paper, latest available version. We have chosen `H2Lib`, since it is under active development and an Open Source library. The `H2Lib` library implements an algebra for $\mathcal{H}$ matrices and $\mathcal{H}^2$ matrices. That is, the library allows to construct, add, multiply, factorize, etc. $\mathcal{H}$ and $\mathcal{H}^2$ matrices. Moreover, it contains modules for the solution of problems discretized by the boundary element method. Recently, support for a GPU-accelerated $\mathcal{H}^2$ matrix setup for boundary element method problems was added [BC15], as discussed in Section 6.1. Finally, `H2Lib` supports shared-memory parallelism based on OpenMP.

As argued before, the comparison of our GPU implementation, which only implements the $\mathcal{H}$ matrix-vector product, with this feature-complete shared-memory parallel CPU implementation, which has been specifically optimized for $\mathcal{H}^2$ matrices and boundary element method problems, might be unbalanced. However, we add this comparison to somehow relate our performance results to currently available software in the field.

In our performance benchmarks, we try our best to fit the `H2Lib` implementation to our

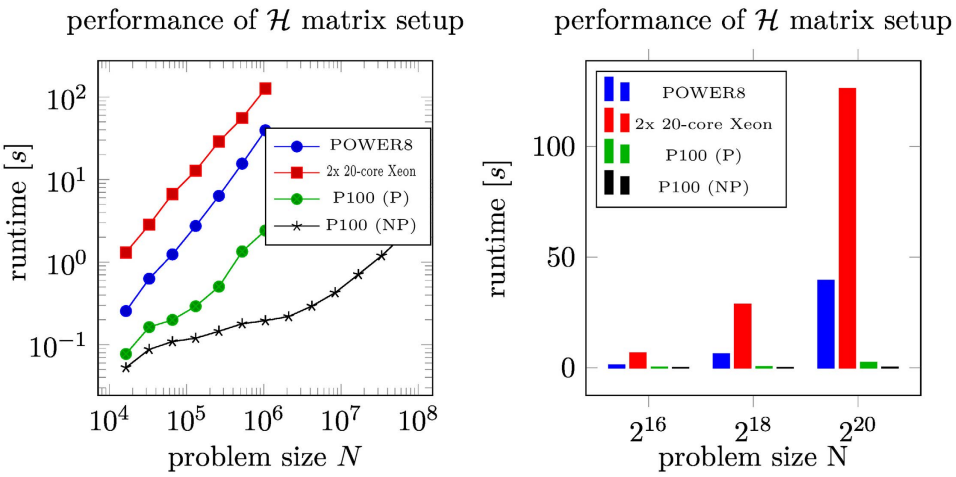


Figure 6.15: We compare the runtime of the $\mathcal{H}$ matrix setup in the *H2Lib* (including the computation of the ACA and all dense sub-blocks) with the setup in the `hmglib` library without *(NP)* and with *(P)* pre-computing the ACA. The GPU-based implementation outperforms the OpenMP-parallel CPU-based implementation by a factor of 630 (NP) / 50 (P) on the almost equally-priced Xeon hardware.

GPU implementation, even if this means that we have to extend the `H2Lib` for this. In fact, we added the ability to do ACA for a fixed rank k, which was not available in the library, before. Moreover, we had to add assembly routines for the system matrices required in our examples. Note here that we parallelized the new assembly routines by OpenMP identical to the already existing assembly routines in the `H2Lib`. On the IBM POWER8 platform, `H2Lib` is compiled with `gcc` 4.8.5, OpenMP support and the usual optimizations and linked against the, at time of writing this article, latest available version of `OpenBLAS` (0.2.20). On the Intel architecture, `gcc` 4.8.5 is replaced by the most recent version 8.1.0. All other settings are kept.

We start the comparison with a benchmark of the $\mathcal{H}$ matrix construction or setup phase. In case of the `H2Lib` this construction phase contains the spatial data structure setup, the block cluster tree traversal, the pre-computation of all low-rank factors and the assembly of all dense sub-blocks of the $\mathcal{H}$ matrix. We choose $\eta = 1.5$ and $C_{leaf} = 128$. On the other hand, we choose $\eta = 1.5$, $C_{leaf} = 2048$, $bs_{dense} = 2^{27}$ and $bs_{ACA} = 2^{25}$ in the GPU implementation and analyse the construction phase including pre-computation *(P)* of the ACA factors or without *(NP)* such a pre-computation. Note that the leaf size C_{leaf} has a significant impact on the performance on the method, cf. Section 6.5.6. Therefore, we adapt it for the different architectures for best possible performance. All results in this paragraph are computed for a fixed rank of $k = 16$ and dimension $d = 2$.

Figure 6.15 gives the result for the first comparison. On the left-hand side, runtimes of the setup phase are given for growing problem size. The diagram on the right-hand side directly compares the results on the different architectures for fixed problem sizes. Except of the GPU benchmark with pre-computing, all other benchmarks are stopped for $N = 2^{20}$ due to excessive runtime or limited memory. Let us remember here that we compare a single-

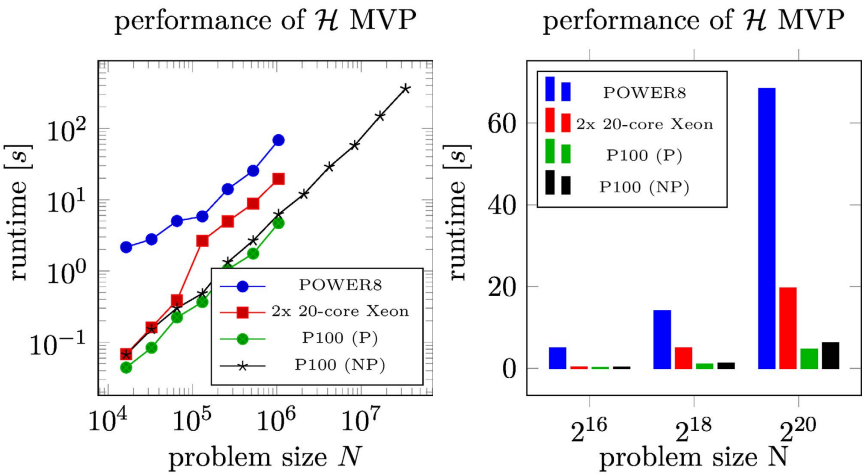


Figure 6.16: The GPU-based $\mathcal{H}$ matrix-vector product outperforms the OpenMP-parallel CPU-based $\mathcal{H}$ matrix vector product by a factor of 4.2 on the roughly equally priced Xeon hardware, when pre-computing the ACA factors.

GPU implementation to an OpenMP-parallel code running on 40 physical / 80 logical cores on the Intel Xeon compute node and to an OpenMP-parallel code running on 20 physical / 160 logical cores on the IBM POWER8 compute node. In case of the largest common problem size, i.e. $N = 2^{20}$ ($\sim$ one million points), the shared-memory parallel implementation requires 126.1 seconds on the Xeon system and 39.5 seconds on the POWER8 system.[6] On the other hand, the GPU implementation only needs 2.4 seconds with pre-computing and 0.2 seconds without pre-computing. That is, it is by about a factor 630 / 50 (without / with pre-computing) faster than the parallel code on the Xeon system, i.e. comparing roughly equally priced hardware. Also, it is by a factor of 200 / 16 faster compared to the POWER8 system, of which the pricing is unknown to the author. However, note again that the setup phase on CPU also pre-computes the dense matrix sub-blocks.

Our second comparative study targets the $\mathcal{H}$ matrix-vector product. It is done with the same parameters as before. The results are given in Fig. 6.16. We observe a clear performance improvement of the larger GPU-based runs against the CPU-based results. Comparing again the results for $N = 2^{20}$, we see a runtime of about 19.6 seconds on the Xeon system and 68.4 seconds on the POWER8 system. In contrast, we get a runtime of 6.2 seconds without ACA pre-computing and an improvement by about 30 % to 4.7 seconds with pre-computing on GPU. This is a remarkable performance improvement by a factor of 3.2 / 4.2 on GPU over the OpenMP-parallel running on roughly equally priced Xeon hardware. On the POWER8 system, the gain is even more pronounced with a factor of 11 / 14.6. At this point, we still have to keep in mind that the CPU-based code assembles and stores all dense matrix sub-blocks of the approximated matrix, beforehand, while `hmglib` re-computes these on-the-fly due to memory limitations. Moreover, we still see some room for performance improvements of the GPU implementation.

[6] We did not observe perfect scalability of the OpenMP parallel code on the CPU systems. This should be kept in mind, when comparing CPU to GPU results.

Overall, we conclude that a perfectly fair comparison is hardly possible. CPU-based implementations rely much more on pre-computation and therefore might have a slight advantage for the $\mathcal{H}$ matrix-vector product, while being much slower in the setup phase. The new GPU-based implementation tries to balance the strong memory restrictions of GPUs with a general performance improvement. Based on the raw numbers of the equally priced Xeon hardware CPU to GPU comparison, the GPU code outperforms the OpenMP-parallel CPU code by a factor of 630 / 50 (without / with pre-computing) for the setup and by a factor of 3.2 / 4.2 (without / with pre-computing) in the matrix-vector product. On the POWER8 system these are factors of 200 / 16 in the setup and 11 / 14.6 in the matrix-vector product.

6.6 Summary

This work considered the reformulation of algorithms in the construction and matrix-vector product of $\mathcal{H}$ matrices for many-core parallelism. Core techniques to get fast many-core parallel performance were a parallel spatial data structure based on space filling curves, parallel tree traversal and batching of many small, non-equally sized compute tasks. Using these techniques, we designed new algorithms for many-core parallel $\mathcal{H}$ matrices. These algorithms were transferred to a reference implementation on a GPU, which results in the GPU $\mathcal{H}$ matrix library `hmglib`. Our numerical results showed that the designed algorithms lead to a fast GPU implementation. Comparing the new library running on one Tesla P100 SXM2 GPU to an OpenMP-parallel version of the `H2Lib` library running on roughly similar priced Intel Xeon hardware, we achieve more than a factor of 50 performance improvement in the $\mathcal{H}$ matrix construction and about a factor of 4.2 in the matrix-vector product (using ACA pre-computing). Note however that comparing the libraries is somewhat difficult. Nevertheless, we tried our best to keep this comparison fair.

In the future, our new algorithms shall be extended to the use in a distributed-memory, thus e.g. multi-GPU, context. This, however, involves to build an appropriate load balancing for the work distribution of ACA computations and dense matrix-vector products on an entire cluster of compute nodes equipped with many-core hardware. Moreover, the heterogeneous nature, i.e. the existence of powerful CPUs and many-core devices, of current compute clusters could be rather simply included by transferring a part of the numerical linear algebra work queue to the CPUs. Thereby, an even higher performance could be achieved on these systems.

6.7 Appendix: Batched bounding box computation

As part of the traversal of the block cluster tree, we have to evaluate the admissibility condition (6.3) for index blocks $\tau \times \sigma$ involving the bounding boxes of τ and σ in each node. In the following, we will discuss an algorithm to concurrently compute the bounding boxes for clusters τ, σ in all nodes on a given level l of the block cluster tree. The algorithm is based on batching, cf. Section 6.4.3.

We collect the set of nodes on a level l of the block cluster tree, i.e. $V_{I\times I}(l)$, in the array `node_data` of length $|V_{I\times I}(l)|$ composed of structs `work_item` and have the input points $\mathcal{Y}$ in an instance of struct `point_set`, cf. Section 6.4.1. As simplification, we only consider the

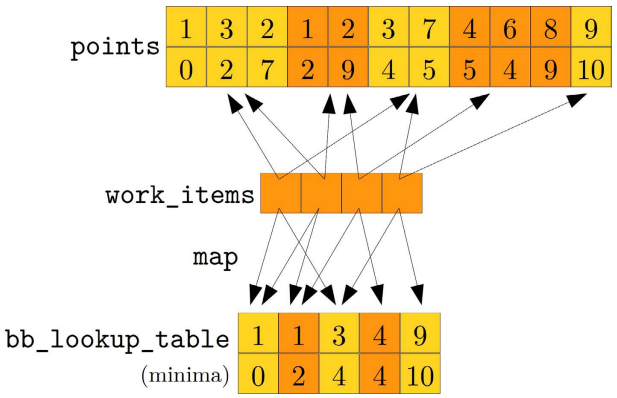


Figure 6.17: The boundary box computation is sped up by pre-computing bounding boxes for each cluster, once. They are stored in `bb_lookup_table` and accessed via a map between work items and the lookup table.

concurrent computation of the bounding boxes for one cluster set, e.g. τ, in each node / block cluster.

Note that standard implementations of $\mathcal{H}$ matrix algorithms first build a cluster tree and then construct the block cluster tree. In that case, it is natural to compute the bounding boxes of each cluster in the cluster tree construction. We here, however, have chosen a different approach. That is, we construct the block cluster tree without first building a cluster tree. This is efficient with respect to the clustering, since the Morton ordering of the point sets makes clustering cheap. Moreover, we prefer to do just one tree traversal (for the block cluster tree) instead of several tree traversals (for two cluster trees and the block cluster tree) to maximize the utilization of the many-core processor within a single tree traversal. However, this comes at the price that the bounding box computation becomes part of the block cluster tree traversal, which is quite unusual. To avoid to compute bounding boxes for a single cluster many times, we divide the bounding box calculation in a pre-processing step and the bounding box assignment. In the pre-processing step, we extract from all block clusters the list of clusters. This list potentially contains multiple copies of clusters. We then unify this list to create a unique list of clusters. For each *unique cluster* in that unified list, we compute the bounding boxes and store them in a lookup table. This corresponds to computing the bounding boxes on the cluster tree. In the bounding box assignment step, we use a lookup table to assign the pre-computed bounding boxes to the clusters in the block cluster tree. Thereby, we avoid recomputing bounding boxes for identical clusters.

Since the algorithmic demanding step is the pre-processing step, we stick to the description of this step. Here, we first identify the set of unique clusters, as defined before. We then create a lookup table `bb_lookup_table` storing for each unique cluster the bounding box information. In addition, we need a map from a node in `node_data` to the entry in the lookup table. Figure 6.17 exemplifies this idea.

Algorithm 14 describes our approach to compute the entries of the lookup table `bb_lookup_table`. Function COMPUTE_BOUNDING_BOX_LOOKUP_TABLE gets as input the coordinate array `coords` of the input point set $\mathcal{Y}$, the nodes $V_{I\times I}(l)$ on level l in `node_data`, and

Algorithm 14 Compute bounding box lookup table

```
procedure COMPUTE_BOUNDING_BOX_LOOKUP_TABLE(node_data, coords, l, |V_{I×I}(l)|)
    (lower_bounds, upper_bounds) ← GET_INDEX_BOUNDS(node_data)
    STABLE_SORT(lower_bounds)
    STABLE_SORT(upper_bounds)
    UNIQUE(lower_bounds, unique_lower_bounds)
    UNIQUE(upper_bounds, unique_upper_bounds)
    lookup_table_size ← |lower_bounds|
    batch_bounds ← (unique_lower_bounds, unique_upper_bounds)
    SEQUENCE(unique_set_indices, lookup_table_size, 1)
    INIT<|𝒴|>(batch_keys,0)
    batch_keys ← CREATE_KEYS(batch_bounds, unique_set_indices, |𝒴|, lookup_table_size)
    (coord_maxima, output_keys) ← REDUCE_BY_KEY(coords, batch_keys,maximum)
    (coord_minima, output_keys) ← REDUCE_BY_KEY(coords, batch_keys,minimum)
    REMOVE_BY_KEY(coord_maxima, output_keys, 0)  ▷ Remove invalid compute results
    REMOVE_BY_KEY(coord_minima, output_keys, 0)
    bb_lookup_table ← (coord_minima, coord_maxima)
    return bb_lookup_table
```

Algorithm 15 Generator for map to bounding box table

```
procedure CREATE_MAP_FOR_BOUNDING_BOXES(node_data, l, |V_{I×I}(l)|)
    (lower_bounds, upper_bounds) ← GET_INDEX_BOUNDS(node_data)
    SEQUENCE<|V_{I×I}(l)|>(permutation, |V_{I×I}(l)|)   ▷ permutation ← (0, 1, ..., |V_{I×I}(l)|)
    STABLE_SORT_BY_KEY(permutation, lower_bounds)
    INIT<|V_{I×I}(l)|>(map, 0)
    SET_BOUNDS_FOR_MAP<|V_{I×I}(l)|>(map, lower_bounds)
    INCLUSIVE_SCAN(map, map, 0, |V_{I×I}(l)|)
    PERMUTE_MAP<|V_{I×I}(l)|>(map, permutation)
    return map
```

further size information. First, the lower index bounds $i_{l,1}$ and upper index bounds $i_{u,1}$ are extracted from each node and stored in arrays `lower_index_bounds` and `upper_index_bounds`. By construction, the (block) cluster tree traversal based on Z-order curves only creates clusters that do not overlap in the point set array and that, for a given lower index bound, will always have the same upper bound. Therefore, we can use parallel sorting and unification methods to identify the set of unique clusters. The unique clusters are collected (by their lower and upper index bounds) in `unique_lower_index_bounds` and `unique_upper_index_bounds`. The final step is to compute the coordinate minima and maxima in each cluster. This step follows the idea of batching, cf. Section 6.4.3. The *batched array* is the array of coordinates. The bounds for the batches are given by the unique lower and upper index bounds and the keys for the batches are the sequence of numbers $\{1, 2, \ldots\}$. Results in the batched computation that are associated to points in $\mathcal{Y}$ and not being part of any cluster are finally removed by removing all batched compute results associated to the key 0.

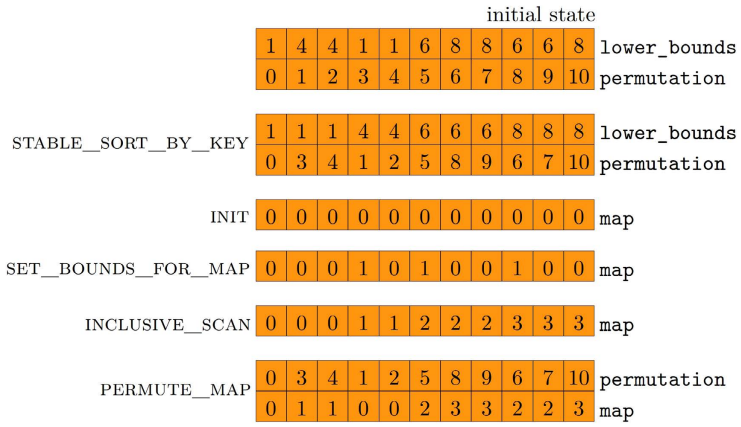


Figure 6.18: Creating the map between a work item and an entry in the lookup table requires sorting, a compute kernel for bounds assignment, a scan operation and a permutation operation.

Our approach to compute the map between the nodes in `node_data` and the lookup table is summarized in Algorithm 15. Again, we first get the lower and upper index bounds. Then, without loss of generality, we sort the lower bounds of the clusters and keep the applied permutation in `permutation`. Next, we create a global array `map` of length $|V_{I\times I}(l)|$ and initialize it to "0". The parallel kernel SET__BOUNDS__FOR__MAP of $|V_{I\times I}(l)|$ threads then sets a "1" in `map` wherever there are two different subsequent entries in the sorted `lower_bounds`. By an inclusive scan on `map`, we create growing indices in `map` marking identical entries in `lower_bounds`. The result is exemplified in Fig. 6.18. We finally permute back `map` by kernel `permutation` with $|V_{I\times I}(l)|$ threads leading to the required map.

Acknowledgements

The author would like to thank the anonymous referees for their enlightening remarks. This work is funded by the Swiss National Science Foundation (SNF) under project number 407540_167186. Furthermore, code developments tasks in this research were done on resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. The IBM POWER8 system with the NVIDIA Tesla P100 SXM2 used in the benchmarks for this research was donated by the *NVIDIA PSG Cluster*. All funding and support is gratefully acknowledged.

References

[ABC+14] E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner, and T. Takahashi. Task-based FMM for multicore architectures. *SIAM Journal on Scientific Computing*,

36(1):C66–C93, 2014.

[AHTD17] A. Abdelfattah, A. Haidar, S. Tomov, and J. Dongarra. Novel HPC techniques to batch execution of many variable size BLAS computations on GPUs. In *Proceedings of the International Conference on Supercomputing*, ICS '17, pages 5:1–5:10, New York, NY, USA, 2017. ACM.

[BÏ7] S. Börm. `H2Lib`, a library for hierarchical matrices, 2017.

[BC15] S. Börm and S. Christophersen. Approximation of BEM matrices using GPGPUs. *ArXiv e-prints*, October 2015.

[Beb] M. Bebendorf. `AHMED` Another software library on hierarchical matrices for elliptic differential equations.

[Beb08] M. Bebendorf. *Hierarchical Matrices - A Means to Efficiently Solve Elliptic Boundary Value Problems*, volume 63 of *Lecture Notes in Computational Science and Engineering*. Springer, 2008.

[BET99] M. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtrees and quality triangulations. *International Journal of Computational Geometry & Applications*, 09(06):517–532, 1999.

[BGH03] S. Börm, L. Grasedyck, and W. Hackbusch. Introduction to hierarchical matrices with applications. *Engineering analysis with boundary elements*, 27(5):405–422, 2003.

[BH11] N. Bell and J. Hoberock. Thrust: A productivity-oriented library for CUDA. *GPU computing gems Jade edition*, 2:359–371, 2011.

[BK09] M. Bebendorf and S. Kunis. Recompression techniques for adaptive cross approximation. *J. Integral Equations Applications*, 21(3):331–357, 09 2009.

[BLL+] W. Boukaram, H. Ltaief, A. Litvinenko, A. Abdelfattah, and D. E. Keyes. Accelerating matrix-vector multiplication on hierarchical matrices using graphical processing units.

[Bör04] S. Börm. $\mathcal{H}^2$-matrices - Multilevel methods for the approximation of integral operators. *Computing and Visualization in Science*, 7(3):173–181, Oct 2004.

[BR03] M. Bebendorf and S. Rjasanow. Adaptive low-rank approximation of collocation matrices. *Computing*, 70(1):1–24, 2003.

[BTLK17] W. H. Boukaram, G. Turkiyyah, H. Ltaief, and D. E. Keyes. Batched QR and SVD Algorithms on GPUs with Applications in Hierarchical Matrix Compression. *ArXiv e-prints*, July 2017.

[CKL] A. Charara, D. E. Keyes, and H. Ltaief. Batched triangular dense linear algebra kernels for very small matrix sizes on GPUs. *ACM Transactions on Mathematical Software*.

[Fas07] G. F. Fasshauer. *Meshfree Approximation Methods with MATLAB*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2007.

[GKLB08] L. Grasedyck, R. Kriemann, and S. Le Borne. Parallel black box-LU preconditioning for elliptic boundary value problems. *Computing and Visualization in Science*, 11(4):273–291, 2008.

[GLR+16] P. Ghysels, X. S. Li, F. Rouet, S. Williams, and A. Napov. An efficient multicore implementation of a novel HSS-structured multifrontal solver using randomized sampling. *SIAM J. Scientific Computing*, 38(5), 2016.

[GPM11] K. Garanzha, J. Pantaleoni, and D. McAllister. Simpler and faster HLBVH with work queues. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, pages 59–64, New York, NY, USA, 2011. ACM.

[GR97] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta numerica*, 6:229–269, 1997.

[Hac15] W. Hackbusch. *Hierarchical matrices : Algorithms and Analysis*, volume 49 of *Springer series in computational mathematics*. Springer, Berlin, 2015.

[Hac16] W. Hackbusch. Survey on the technique of hierarchical matrices. *Vietnam Journal of Mathematics*, 44(1):71–101, 2016.

[HB02] W. Hackbusch and S. Börm. $\mathcal{H}^2$-matrix approximation of integral operators by interpolation. *Applied numerical mathematics*, 43(1-2):129–143, 2002.

[HKS00] W. Hackbusch, B. Khoromskij, and S. A. Sauter. On $\mathcal{H}^2$-matrices. In *Lectures on Applied Mathematics: Proceedings of the Symposium Organized by the Sonderforschungsbereich 438 on the Occasion of Karl-Heinz Hoffmanns 60th Birthday, Munich, June 30–July 1, 1999*, page 9. Springer Science & Business Media, 2000.

[HN89] W. Hackbusch and Z. P. Nowak. On the fast matrix multiplication in the boundary element method by panel clustering. *Numerische Mathematik*, 54(4):463–491, 1989.

[Kri05] R. Kriemann. Parallel $\mathcal{H}$-matrix arithmetics on shared memory systems. *Computing*, 74(3):273–297, 2005.

[Kri13] R. Kriemann. $\mathcal{H}$-LU factorization on many-core systems. *Comput. Vis. Sci.*, 16(3):105–117, June 2013.

[Kri17] R. Kriemann. $\mathcal{H}$-Lib$^{\text{pro}}$ (website), 2017.

[LGS+09] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH construction on GPUs. *Computer Graphics Forum*, 28(2):375–384, 2009.

[MGG12] D. Merrill, M. Garland, and A. Grimshaw. Scalable GPU graph traversal. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 117–128, New York, NY, USA, 2012. ACM.

[Mor66] G. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical Report Ottawa, Ontario, Canada, 1966.

[MXYB16] W. B. March, B. Xiao, C. Yu, and G. Biros. ASKIT: An efficient, parallel library for high-dimensional kernel summations. *SIAM Journal on Scientific Computing*, 38:S720–S749, 01 2016.

[Pou] J. Poulson. `DMHM` - Distributed-Memory Hierarchical Matrices.

[RLGN16] F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov. A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization. *ACM Trans. Math. Softw.*, 42(4):27:1–27:35, June 2016.

[RW05] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[SDC07] Z. Sheng, P. Dewilde, and S. Chandrasekaran. *Algorithms to Solve Hierarchically Semi-separable Systems*, pages 255–294. Birkhäuser Basel, Basel, 2007.

[Szu16] J. Szuppe. Boost.Compute: A parallel computing library for C++ based on OpenCL. In *Proceedings of the 4th International Workshop on OpenCL*, IWOCL '16, pages 15:1–15:39, New York, NY, USA, 2016. ACM.

[Val90] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[Vov13] V. Vovk. *Kernel Ridge Regression*, pages 105–116. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[Wen04] H. Wendland. *Scattered Data Approximation*. Cambridge University Press, 2004.

[YAM+15] P. Yalamanchili, U. Arshad, Z. Mohammed, P. Garigipati, P. Entschev, B. Kloppenborg, J. Malcolm, and J. Melonakos. ArrayFire - A high performance software library for parallel computing with an easy-to-use API, 2015.

[YB13] R. Yokota and L. Barba. FMM-based vortex method for simulation of isotropic turbulence on GPUs, compared with a spectral method. *Computers & Fluids*, 80:17 – 27, 2013.

[YBK10] R. Yokota, L. Barba, and M. G. Knepley. PetRBF A parallel O(N) algorithm for radial basis function interpolation with Gaussians. *Computer Methods in Applied Mechanics and Engineering*, 199(25):1793 – 1804, 2010.

[Zas17] P. Zaspel. `MPLA` - Massively Parallel Linear Algebra, 2017.

[Zas18] P. Zaspel. `hmglib` - Hierarchical matrices on GPU(s) library, 2018.

7 A scalable $\mathcal{H}$-matrix approach for the solution of boundary integral equations on multi-GPU clusters

7.1 Introduction

The numerical solution of boundary integral equations is an important task in applications from science and engineering such as electric field computations, electromagnetism, acoustic scattering, or fluid mechanics [HW08]. Boundary integral equations arise typically from the reformulation of *boundary value problems* with constant coefficients. In many cases, such a reformulation is advantageous, since a discretization of a boundary integral equation requires the introduction of degrees of freedom just *on the boundary*, while the original partial differential equation needs to be discretized *in the full domain*, which might be unbounded in case of exterior boundary value problems.

Besides the collocation method, the standard approach for discretizing and solving boundary integral equations is based on a Galerkin discretization which amounts to the *boundary element method (BEM)*, see [GKW03, SS11, Ste08] for example. BEM applies standard techniques known from the finite element method (FEM) to the boundary integral equation case. Since the kernel of a boundary integral operator is usually not compactly supported and singular at the diagonal, the stiffness matrix of a BEM discretization is densely populated and computationally expensive to compute. To overcome the cubic complexity of direct factorization approaches, iterative solvers with fast approximate matrix-vector products are used to solve the system of linear equations. Candidates for matrix approximations are the panel clustering [HN89], the fast multipole method [GR97], hierarchical ($\mathcal{H}$) matrices [Beb08, BGH03, Hac15, Hac16] or $\mathcal{H}^2$ matrices [Bör04, HB02, HKS00]. We here focus on $\mathcal{H}$-matrices, since these are a widely use approach and, together with adaptive cross approximation (ACA) [BR03], allow for a purely algebraic construction of the matrix approximation, facilitating its use in real-world applications. For a given fixed accuracy, it is possible to show that the approximate matrix-vector product of $\mathcal{H}$-matrices can be done in $O(N \log N)$ operations.

The objective of this work is to solve large-scale BEM problems by the hierarchical matrix approach. In fact, we aim for solving systems of linear equations from BEM discretizations with hundreds of thousands or millions of unknowns. Such problem sizes arise if the underlying geometry is either very complex or if a solution with a small numerical error is required.

We observe two difficulties when it comes to the solution of large-scale BEM problems. First, the computational runtime becomes excessively large. Second, the required memory is considerable. While the first problem can be addressed by a parallelization of a hierarchical matrix library on a single compute node with high amounts of memory, the second problem can only be fixed by a *distributed-memory* parallelization of the hierarchical matrix approach.

Our conclusion is to work on a distributed-memory parallel implementation for $\mathcal{H}$-matrices. In particular, we apply and extend the many-core parallel Open Source library `hmglib` [Zas17, Zas18] in order to treat BEM-type problems in a distributed-memory parallel way.

`hmglib` is a many-core parallel library allowing to set up and apply $\mathcal{H}$-matrices on a single *graphics processing unit* (GPU). It has been originally developed in the context of the approximation of system matrices from kernel collocation or kernel ridge regression, where it showed decent performance improvements over a parallel $\mathcal{H}$-matrix implementation on standard processors (CPUs). As part of the present work, `hmglib` has been extended such that it can be applied to arbitrary application codes with dense system matrices, as long as the codes provide a means to evaluate matrix entries and the geometric location of the involved degrees of freedom. That is, `hmglib` now provides a general interface e.g. for BEM codes. Moreover, and much more important, the library has been extended such that it is now able to run on the GPUs of a distributed-memory parallel cluster of GPUs.[1] This allows to scale the solution of BEM problems on up to millions of unknowns. While the original implementation of `hmglib` in [Zas17, Zas18] could only pre-compute low-rank blocks, the new implementation is further able to pre-compute and store the (non-admissible) dense matrix blocks in (GPU) memory. This is crucial for BEM applications.

Note that there is a series of related CPU libraries for parallel hierarchical matrices. $\mathcal{H}$-Lib^{pro} [BGH03, GKLB08, Kri05, Kri17] is a commercial shared-memory parallel library with limited distributed-memory support. `AHMED` (Another software library on hierarchical matrices for elliptic differential equations) [Beb] and `DMHM` (Distributed-Memory Hierarchical Matrices) [Pou] provide distributed-memory support on CPUs. `H2Lib` [BÏ17] is shared-memory parallel. In the related field of Hierarchically Semi-Separable (HSS) matrices [SDC07], the software `STRUMPACK` [GLR$^+$16, RLGN16] is shared- and distributed-memory parallel. In context of many-core processors, i.e. GPUs and e.g. Intel Xeon Phi, there is some recent work. An extension to the `H2Lib` [BC15] allows to accelerate the quadrature in a $\mathcal{H}^2$ matrix method for BEM by GPUs. A similar approach is used in the BEM library `Bempp` [SBA$^+$15, VBD17]. Furthermore, [Kri13] discusses a many-core parallel LU-factorization for $\mathcal{H}$-matrices on a Xeon Phi device. `BEM4I` [KMMZ18, MZ18] provides a BEM library with ACA running on clusters of multi-/many-core hardware by Intel based on an MPI, OpenMP and vectorization parallelization. In [OYIY18], the $\mathcal{H}$-matrix vector product (without setup) has been parallelized on a single GPU and on Intel processors. The new *tile low rank* (TLR) format is used in `HiCMA` a library for low-rank Cholesky factorizations on clusters of multi-core and many-core hardware running on Intel hardware [ALM$^+$18] and with the main application of Matérn-type covarice matrices. By some of the authors of [ALM$^+$18], further work has been carried out, which focused on batched dense linear algebra kernels [CKL17] on a single GPU, batched QR and SVD algorithms [BTLK18] on GPUs and a batched TLR GEMM operation on a single GPU [CKL18]. However, to the best of the authors' knowledge, we here discuss the first fully GPU-based distributed-memory parallel hierarchical matrix Open Source library using the traditional $\mathcal{H}$-matrix format and adaptive cross approximation being applied to BEM problems.[2]

[1] The authors do not stick to a GPU+CPU (i.e. GPU acceleration) approach although this approach would better reflect currently available GPU-accelerated HPC installations. This decision has been made since the authors want to tackle the research questions that are connected to large to extreme processing core counts, which are present in case of a GPU-only implementation.

[2] Note that during the review process of this work, [YAI$^+$18] was published. It discusses the (multi-) GPU-

To be able to apply our parallel library to a (large-scale) BEM model problem, we further parallelized an existing sequential CPU code for the solution of elliptic problems by the single-layer potential ansatz with piecewise constant basis functions on GPU and coupled that code to `hmglib`. Thereby, we will be able to show that we can solve large-scale BEM problems in the range of millions of unknowns with a descent strong scaling beyond 68 percent strong scaling efficiency on 1024 GPUs of *Titan* at Oak Ridge National Lab (starting from 128 GPUs due to memory limitations).

The remainder of this work is structured as follows. In Section 7.2, we introduce the mathematical background of boundary integral equations, the boundary element method and the hierarchical matrix approach. Section 7.3 briefly reviews the computational details of `hmglib` and introduces the new general application interface, the dense block storage and multi-GPU parallelization. Numerical results and parallel scalability studies are discussed for a model application and a model code in Section 7.4. We finish by conclusions in Section 7.5.

7.2 Mathematical background

7.2.1 Boundary integral equations

Shall $\Omega \subset \mathbb{R}^d$ with $d = 3$ be a Lipschitz domain and $\Gamma := \partial\Omega$ its surface. We aim at solving boundary integral equations of type

$$(\mathcal{A}u)(\boldsymbol{x}) := \int_\Gamma G(\boldsymbol{x}, \boldsymbol{x}')u(\boldsymbol{x}')\mathrm{d}\sigma_{\boldsymbol{x}'} = f(\boldsymbol{x})\,, \quad \boldsymbol{x} \in \Gamma\,, \tag{7.1}$$

where $\mathcal{A}$ is supposed to be an invertible boundary integral operator. We assume that $\mathcal{A}$ is a continuous and elliptic operator of order $2q$, which means that it maps from $H^q(\Gamma)$ to $H^{-q}(\Gamma)$. We require the integral kernel $G : \Omega \times \Omega \to \mathbb{R}$ to be *asymptotically smooth*, that is, we require to have constants $C_{as1}, C_{as2} \in \mathbb{R}^{>0}$ such that

$$|\partial_{\boldsymbol{x}}^{\boldsymbol{\alpha}} \partial_{\boldsymbol{x}'}^{\boldsymbol{\beta}} G(\boldsymbol{x}, \boldsymbol{x}')| \leqslant C_{as1} \frac{(|\boldsymbol{\alpha}| + |\boldsymbol{\beta}|)!}{(C_{as2}\|\boldsymbol{x} - \boldsymbol{x}'\|)^{|\boldsymbol{\alpha}|+|\boldsymbol{\beta}|}} |G(\boldsymbol{x}, \boldsymbol{x}')|$$

for arbitrary $\boldsymbol{x}, \boldsymbol{x}' \in \Omega$ with $\boldsymbol{x} \neq \boldsymbol{x}'$ and all multi-indices $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{N}_0^d$. This choice allows for kernel functions with singularities at the *diagonal* $\boldsymbol{x} = \boldsymbol{x}'$, while being smooth away from the diagonal.

Example 7.1. *Boundary integral equations of the above type arise in context of the solution of the Laplace equation*

$$\Delta U = 0 \text{ in } \Omega, \quad U = f \text{ on } \Gamma,$$

where $U \in H^1(\Omega)$ is the solution for given Dirichlet data $f \in H^{1/2}(\Gamma)$. Since we know the

parallelization of an iterative $\mathcal{H}$-matrix BiCGStab solver by means of batched MAGMA routines and MPI. This GPU-acceleration work focuses only on the solver (i.e. not the matrix assembly or the $\mathcal{H}$-matrix setup and therefore discusses a sub-set of the work discussed here.

fundamental solution of the Laplace operator, we can make the the single-layer potential ansatz

$$U(\boldsymbol{x}) = \int_\Gamma \frac{u(\boldsymbol{x}')}{4\pi\|\boldsymbol{x}-\boldsymbol{x}'\|_2} d\sigma_{\boldsymbol{x}'} = \tilde{\mathcal{S}}u(\boldsymbol{x})\,, \quad \boldsymbol{x}\in\Omega\,. \tag{7.2}$$

That is, we describe the solution of the Laplace equation by means of the unknown density $u \in H^{-1/2}(\Gamma)$*. Since the single-layer potential is continuous in the whole space* $\mathbb{R}^d$*, the density is obtained by solving the boundary integral equation*

$$(\mathcal{S}u)(\boldsymbol{x}) = \int_\Gamma \frac{u(\boldsymbol{x}')}{4\pi\|\boldsymbol{x}-\boldsymbol{x}'\|_2} d\sigma_{\boldsymbol{x}'} = f(\boldsymbol{x})\,, \quad \boldsymbol{x}\in\Gamma\,, \tag{7.3}$$

where $\mathcal{S} : H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma)$ *is the single-layer operator and* f *the Dirichlet data of the Laplace equation. It can be shown that the kernel function* $\frac{1}{4\pi\|\boldsymbol{x}-\boldsymbol{x}'\|_2}$ *is asymptotically smooth and that* $\mathcal{S}$ *is continuous and continuously invertible.* △

7.2.2 Galerkin BEM discretization

To solve (7.1) by the boundary element method (BEM), we first bring the equation in its variational form: Find $u \in V(= H^q(\Gamma))$, such that

$$\int_\Gamma\int_\Gamma G(\boldsymbol{x},\boldsymbol{x}')u(\boldsymbol{x}')v(\boldsymbol{x})\mathrm{d}\sigma_{\boldsymbol{x}'}\mathrm{d}\sigma_{\boldsymbol{x}} = \int_\Gamma f(\boldsymbol{x})v(\boldsymbol{x})\mathrm{d}\sigma_{\boldsymbol{x}} \text{ for all } v\in V\,.$$

We discretize it by introducing an approximation of the boundary Γ by surface elements

$$\mathcal{T}_h := \{T_1,\dots,T_M\}$$

of size $O(h)$. The elements T_i are usually chosen as planar triangles, cf. [GKW03, Ste08, SS11]. Nonetheless, parametric representations of the surface have recently been become quite popular. Then, $\mathcal{T}_h$ would be a structured quadrangulation and T_i a curved quadrangle, cf. [HR10, MZBF15, DHK$^+$18].

The elements induce a set of nodes

$$X_h := \{\boldsymbol{x}_1,\dots,\boldsymbol{x}_N\}\,.$$

We associate to each node $\boldsymbol{x}_i$ a locally supported piecewise polynomial φ_i of order p leading to a finite-dimensional trial space

$$V_h = \{\varphi_1,\dots,\varphi_N\} \subset V\,.$$

Then, we look for an approximate solution $u_h \in V_h$, such that

$$\int_\Gamma\int_\Gamma G(\boldsymbol{x},\boldsymbol{x}')u_h(\boldsymbol{x}')v_h(\boldsymbol{x})\mathrm{d}\sigma_{\boldsymbol{x}'}\mathrm{d}\sigma_{\boldsymbol{x}} = \int_\Gamma f(\boldsymbol{x})v_h(\boldsymbol{x})\mathrm{d}\sigma_{\boldsymbol{x}} \text{ for all } v_h\in V_h\,.$$

With $u_h(\boldsymbol{x}) := \sum_{i=1}^{N} \alpha_i \varphi_i(\boldsymbol{x})$, we finally have to solve the dense linear system

$$\boldsymbol{A}\boldsymbol{\alpha} = \boldsymbol{f} \tag{7.4}$$

with

$$\boldsymbol{A} = [a_{i,j}]_{i,j=1}^{N}, \quad a_{i,j} = \int_\Gamma \int_\Gamma G(\boldsymbol{x}, \boldsymbol{x}') \varphi_i(\boldsymbol{x}') \varphi_j(\boldsymbol{x}') \mathrm{d}\sigma_{\boldsymbol{x}'} \mathrm{d}\sigma_{\boldsymbol{x}}$$

and

$$\boldsymbol{f} = [f_i]_{i=1}^{N}, \quad f_i = \int_\Gamma f(\boldsymbol{x}) \varphi_i(\boldsymbol{x}) \mathrm{d}\sigma_{\boldsymbol{x}} \,.$$

7.2.3 Hierarchical matrices

We aim at solving (7.4) by an iterative method. To make this tractable for large N, we use an approximate matrix-vector product for the Galerkin system matrix $\boldsymbol{A}$. Our choice is to use the purely algebraic hierarchical matrices [BGH03, Hac15] with adaptive cross approximation [BK09, BR03], leading to $O(N \log N)$ complexity for the matrix-vector product, if we fix the approximation tolerance.

Let us briefly consider the approximation of the Galerkin system matrix $\boldsymbol{A}$ by hierarchical matrices. We first introduce the concept of index sets $I := \{1, \ldots, N\}$, representing the nodes $X_h = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ and basis functions $V_N = \{\varphi_1, \ldots, \varphi_N\}$ on Γ. Thereby we can associate an index tuple (i, j) to a geometric location and to each entry $a_{i,j}$ of the system matrix $\boldsymbol{A}$. We will group these index sets into *clusters* $\tau \subset I$ based on geometrical arguments. The product of two clusters (i.e. a *block cluster*), e.g. $\tau \times \sigma \subset I \times I$, can then be translated to a sub-matrix $\boldsymbol{A}|_{\tau \times \sigma}$ of our Galerkin system matrix $\boldsymbol{A}$.

The core idea of hierarchical matrices relies on the fact that for an asymptotically smooth kernel, the evaluation of $\int_\Gamma \int_\Gamma G(\boldsymbol{x}, \boldsymbol{x}') \varphi_i(\boldsymbol{x}') \varphi_j(\boldsymbol{x}') \mathrm{d}\sigma_{\boldsymbol{x}'} \mathrm{d}\sigma_{\boldsymbol{x}}$ for two geometrically well separated basis functions φ_i, φ_j can be approximated with a controlled, small error. This can be expanded to the *admissibility*, i.e. the approximability, of a whole block cluster (of nodes). A typical admissibility condition for a block cluster $\tau \times \sigma$ is based on the bounding boxes for clusters τ, σ (compare e.g. [Hac15] for alternatives). The bounding box of cluster $\tau \subset I$ is $Q_\tau := \prod_{i=1}^{3} \left[a_\tau^{(i)}, b_\tau^{(i)}\right]$ with $a_\tau^{(i)} := \min_{j \in \tau} x_j^{(i)}$, $b_\tau^{(i)} := \max_{j \in \tau} x_j^{(i)}$ and $\boldsymbol{x}_j := \left(x_j^{(1)}, x_j^{(2)}, x_j^{(3)}\right)^\top$. Then, we can introduce the admissibility condition

$$\min\{\mathrm{diam}(Q_\tau), \mathrm{diam}(Q_\sigma)\} \leqslant \eta \mathrm{dist}(Q_\tau, Q_\sigma) \,, \tag{7.5}$$

where $\eta \in \mathbb{R}^{\geqslant 0}$ balances convergence and algorithmic complexity and $\mathrm{diam}(Q_\tau)$ and $\mathrm{dist}(Q_\tau, Q_\sigma)$ are the diameter and the distance of bounding boxes, respectively, cf. [BGH03].

Clusters τ shall always collect geometrically close nodes. They are collected in a *cluster tree* $\mathcal{T}_I = (\mathcal{V}_I, \gamma)$, which imposes a spatial data structure with a hierarchy on I (or X_N). With $\mathcal{V}_I \subset \mathcal{P}(I)$ being the set of nodes in the tree, i.e. the clusters, γ a mapping $\gamma : \mathcal{V}_I \to \mathcal{P}(\mathcal{V}_I)$ of each cluster to its hierarchical sub-clusters, a cluster tree is given such that

(C1) $\tau \in \mathcal{P}(I) \backslash \{\varnothing\}$, for all $\tau \in \mathcal{V}_I$,

(C2) $\mathrm{root}(\mathcal{T}) = I$,

Algorithm 16 Algorithm to build a block cluster tree

```
procedure BUILD_BLOCK_CLUSTER_TREE(τ × σ, C_leaf)
    if τ × σ is not admissible and |τ| > C_leaf and |σ| > C_leaf then
        γ(τ × σ) ← ∅
        for τ' ∈ γ(τ) do                          ▷ Loop over children in cluster trees.
            for σ' ∈ γ(σ) do
                γ(τ × σ) ← γ(τ × σ) ∪ {τ' × σ'}   ▷ Add new child.
                BUILD_BLOCK_CLUSTER_TREE(τ' × σ', C_leaf)
    else
        γ(τ × σ) ← ∅                               ▷ τ × σ becomes leaf.
```

(C3) if $\tau \in \mathcal{V}_I$ is a leaf, i.e. $\gamma(\tau) = \varnothing$, then $|\tau| \leqslant C_{leaf}$ and

(C4) if $\tau \in \mathcal{V}_I$ is no leaf, then it has exactly two children $\gamma(\tau) = \{\tau_1, \tau_2\}$ and $\tau = \tau_1 \cup \tau_2$.

In *cardinality-based clustering* (CBC), which will be use in this work, we further impose $|\tau_1| \approx |\tau_2|$ in **(C4)**.

With a given cluster tree, we can introduce the *block cluster tree* $\mathcal{T}_{I\times I} = (\mathcal{V}_{I\times I}, \gamma, \mu)$, which builds a hierarchy of block clusters out of the given cluster hierarchy. Here, $\mathcal{V}_{I\times I}$ is the set of nodes / block clusters in the tree and γ maps a block cluster to its children. Algorithm 16 recursively defines the block cluster tree and is launched with $\tau \times \sigma = I \times I$.

The corresponding sub-matries $\boldsymbol{A}|_{\tau\times\sigma} \in \mathbb{R}^{|\tau|\times|\sigma|}$ of admissible block clusters in $\mathcal{T}_{I\times I}$ are approximated by an $\mathcal{R}(k)$ *matrix* $\boldsymbol{R}_{\tau\times\sigma} \in \mathbb{R}^{|\tau|\times|\sigma|}$, a matrix of maximum rank k, which is defined as

$$\boldsymbol{R}_{\tau\times\sigma} = \boldsymbol{U}_{\tau\times\sigma}\boldsymbol{V}_{\tau\times\sigma}^{\top}, \quad \boldsymbol{U}_{\tau\times\sigma} \in \mathbb{R}^{|\tau|\times k}, \quad \boldsymbol{V}_{\tau\times\sigma} \in \mathbb{R}^{|\sigma|\times k}.$$

Matrix-vector products with $\mathcal{R}(k)$ matrices $\boldsymbol{R}_{\tau\times\sigma}$ have a computational complexity of $O\left(r \cdot (|\tau| + |\sigma|)\right)$. As in [Zas17]. we will use the algebraic *adaptive cross approximation* (ACA) [BR03, BK09] with partial pivoting to approximate sub-matrices $\boldsymbol{A}|_{\tau\times\sigma} \in \mathbb{R}^{|\tau|\times|\sigma|}$. ACA can be seen as a pivoted Gauss elimination and constructs a low-rank approximation by successive rank-one updates.

Given a rank $k \in \mathbb{N}$ and a block cluster tree $\mathcal{T}_{I\times I}$, we introduce an $\mathcal{H}$*-matrix* of block-wise rank k as matrix $\boldsymbol{L} \in \mathbb{R}^{|I|\times|I|}$ such that

$$\operatorname{rank}(\boldsymbol{L}|_{\tau\times\sigma}) \leqslant k$$

for all admissible $\tau \times \sigma$. The construction of an $\mathcal{H}$-matrix for a dense matrix is known as *truncation*. Matrix-vector products between $\mathcal{H}$-matrices an vectors are realized by a recursive traversal of the block cluster tree. In each non-admissible leaf, the corresponding (precomputed) full sub-matrix is applied, while in admissible leafs the (pre-computed) low-rank approximation is applied. It has been shown, that specific versions of this approach allow to perform an $\mathcal{H}$-matrix-vector product in complexity $O(k \cdot N \log N)$ [Hac15].

7.3 Scalable parallel $\mathcal{H}$-matrix approach for BEM

Our approximation of the Galerkin matrix by hierarchical matrices is based on the library `hmglib` [Zas17, Zas18]. The present work aims at extending `hmglib` such that it can be used for the multi-GPU parallel solution of large-scale boundary integral equations discretized by the boundary element method. To this end, an abstract code interface, precomputation of dense matrix blocks and a distributed-memory parallelization had to be introduced. In the following, we give a brief overview of the original implementation [Zas17] and discuss the new techniques that have been added to `hmglib`.

Remark on technical details. Note that an in-depth technical description of state-of-the-art GPU-parallel codes requires a lot of technical details, such as memory hierarchies, parallelization models, scheduling, caching, etc. As in [Zas17], we here stick to a much less technically overwhelming discussion. To this end, we categorize parallel work loads either into work loads that can be handled by the use of a large amount of parallel threads working on independent tasks or into work loads that require the use of more complicated algorithms with complex thread interactions, such as reduction operations. While the first type of work loads can be easily parallelized by standard GPU parallelization techniques, i.e. in CUDA kernels, we use existing GPU libraries for the second type of work loads, whenever this is possible.

7.3.1 `hmglib` - A many-core parallel $\mathcal{H}$-matrix library

The Open Source GPU library `hmglib` has originally been developed for the approximation of matrices from *kernel interpolation / collocation* or *kernel ridge regression*. It uses a given single GPU for all work loads involved in the construction and application of a hierarchical matrix, i.e. it is not an accelerated but a solely GPU-based software. To be able to get high performance on GPU, the library uses a parallel traversal of the block cluster tree, space-filling curves (to build the clustering) and the concept of batching for many small similarly-sized work loads. In terms of software and hardware, it requires an *Nvidia* GPU and uses the `CUDA Toolkit`, i.e. CUDA kernels [Hal08] for direct GPU programming, the STL-type algorithm library `Thrust` [HB10] running in parallel on a GPU and the BLAS/LAPACK-type libraries `CUBLAS` and `Magma` [DGH$^+$14, HDT$^+$15].

Block cluster tree traversal. To get a high parallel performance on many-core hardware, it is necessary to express an existing algorithm in a very parallel way. In [Zas17], this has been achieved for the construction and traversal of the block cluster by *level-wise* parallelization of the tree traversal. That is, all entries of a given level of a tree are computed in a many-core parallel fashion, while calculations of offsets in the memory are computed by appropriate parallel *scan* operations. Figure 7.1 outlines this methodology. The red arrows on a given level correspond to the parallel threads that are executed on the GPU. As it becomes obvious, the first few levels of a tree do not lead to a full parallel utilization. However, on higher levels, this limitation is no longer present.

Spatial data structure. While classical implementations of $\mathcal{H}$-matrices use spatial data structures such as *kD-trees* or *quad-/oct-trees*, clustering in `hmglib` is based on *space filling curves*

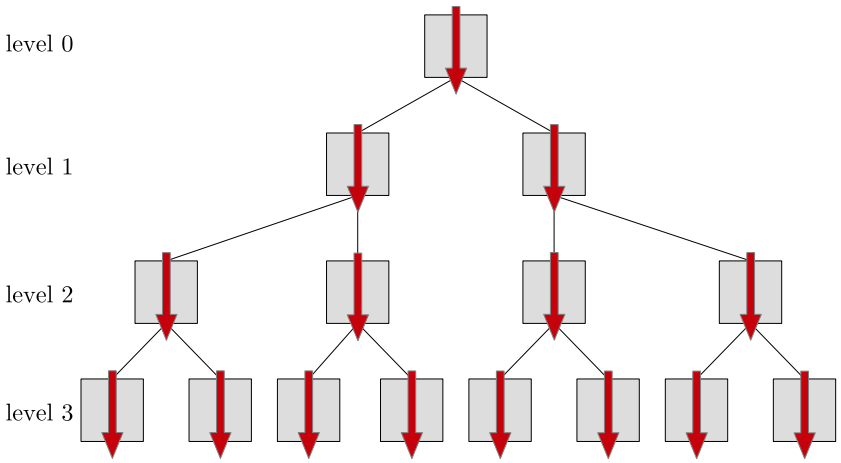


Figure 7.1: In `hmglib`, tree traversal is parallelized in a level-wise fashion. The red arrows on each level correspond to the executed parallel threads.

[LGS+09, Mor66, Zas17]. In particular, a *Morton code* [Mor66] is computed for each node in X_h. This computation is done in a many-core parallel way. After sorting the points in X_h following their Morton codes by a GPU-parallel sorting method, two consecutive nodes in the resulting (sorted) array of nodes are geometrically close. In particular, cardinality-based clustering, cf. Section 7.2.3, can be reduced to simple array decompositions.

Batching. As shown in [Zas17], the highest impact on the GPU-parallel performance of the `hmglib` code is achieved by *batching* of small similarly sized compute work loads into bigger batches of compute work. This is specifically used in `hmglib` in the context of the computation of dense matrix blocks $\boldsymbol{A}_{\tau\times\sigma}$ and low-rank matrix blocks $\boldsymbol{R}_{\tau\times\sigma}$ and in context of the determination of bounding box sizes for the clusters. Figure 7.2 outlines the general strategy. Instead of solving (in parallel) several small problems (e.g. the summation of several numbers), all these operations are batched together in one bigger work load. This allows to achieve a much higher utilization of the GPU and, thus, leads to higher performance. While this strategy is supported in `Thrust` for, e.g., reduction operations by providing index arrays marking the sub-workloads, more complex algorithms such as the adaptive cross approximation in `hmglib` had to be adapted to use this new strategy.

7.3.2 Abstract program interface.

In [Zas17], `hmglib` was just used for system matrices from kernel interpolation / collocation or kernel ridge regression. Such matrices require the evaluation of a very simple kernel function G, which was hard-coded. The new developments in the context of this work start adding an interface for arbitrary application codes that provide custom matrix entries of system matrices that shall be approximated. Besides of standard configuration options for $\mathcal{H}$-matrices, three general *inputs* have to be provided by an application code:

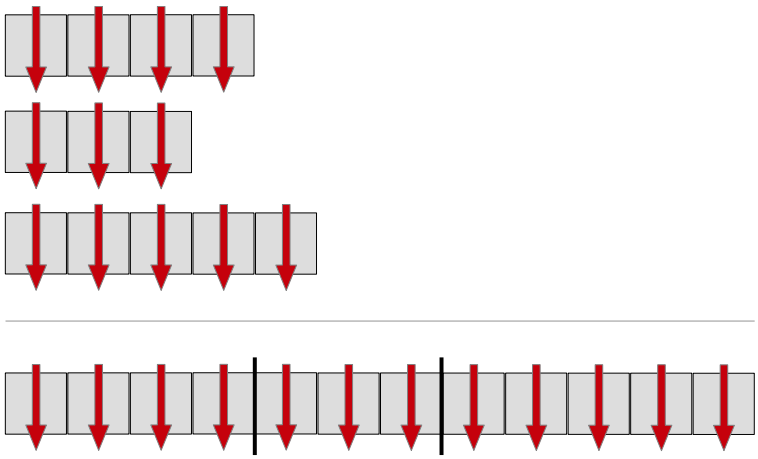


Figure 7.2: Instead of solving several smaller problems in parallel, *batching*, as used in `hmglib`, aims at combining all problems into a single much bigger work load, leading to a higher utilization and higher performance of a GPU.

1. The sets of nodes $X_1 := \{\boldsymbol{x}_1^{(1)}, \ldots, \boldsymbol{x}_{N_1}^{(1)}\}$, $X_2 := \{\boldsymbol{x}_1^{(2)}, \ldots, \boldsymbol{x}_{N_2}^{(2)}\}$. In our BEM application, we have $X_1 = X_2 = X_h$.

2. Functions $\mathrm{idx}_1 : X_1 \to \mathbb{N}$, $\mathrm{idx}_2 : X_2 \to \mathbb{N}$, associating to each node an index. This allows to introduce an `hmglib`- and ordering-independent way to identify the nodes in X_1 and X_2 by the application. In our application, this is simply the index of the nodes.

3. A callback-type function that can be called by `hmglib` in order to evaluate a single entry $a_{i,j}$ of the system matrix $\boldsymbol{A}_{X_1 \times X_2}$ for which the $\mathcal{H}$-matrix shall be constructed. In our application, this is the quadrature routine that computes the matrix entry

$$a_{i,j} = \int_\Gamma \int_\Gamma G(\boldsymbol{x}, \boldsymbol{x}') \varphi_i(\boldsymbol{x}') \varphi_j(\boldsymbol{x}') \mathrm{d}\sigma_{\boldsymbol{x}'} \mathrm{d}\sigma_{\boldsymbol{x}} \,.$$

Technical challenge. It turned out that developing a generalized way to provide a callback function for the matrix entry evaluation is not easy, at least in connection with the CUDA programming language extension provided by the `CUDA Toolkit` 8.0. While it is technically possible to use function pointers to *device functions*, i.e. functions that are executed on *and* launched from GPU, the practical use of these pointers leads to a strongly reduced performance. In order to work around this, we provide a purely virtual abstract *device* basis class with a function `get_matrix_entry`. This class is overwritten by the application that aims to use `hmglib`. At the same time, `hmglib` uses *dynamic polymorphism* to launch `get_matrix_entry` in an application-independent way.

Nonetheless, this solution introduces two difficulties. The first difficulty is in the memory management of the interface *device* class. It has to be instantiated and destroyed from *device* code. Therefore, its use in a library context, in which the interface code is supposed to run on CPU, tends to be rather involved. The second difficulty shows up in the compile and link

process between the library `hmglib` and the application code. Ideally, the aim would be to provide `hmglib` as a shared library. However, in order to be able to use dynamic polymorphism on GPUs in the context of CUDA, it is necessary to put the calling *device* code, the abstract *device* basis class and the overwriting *device* class into the same *compilation unit*. While it is still possible to individually compile the *device* code for the calling code, the basis class and the overwriting class into *device*-only object files, these object files have to be linked by the device code linker before they can be put together with the CPU / *host* code. In practice, this breaks the clear distinction of library and application code. To the best of the authors' knowledge, there is currently no alternative to this approach when using GPUs and *CUDA*.

7.3.3 Pre-computation of matrix blocks.

In [Zas17], some of us discussed the case of $\mathcal{H}$-matrix approximation for collocation matrices. The computational effort to compute the individual system matrix entries in that case was very low. Therefore, it was only considered to pre-compute the low-rank factors for the $\mathcal{R}(k)$-matrices $\boldsymbol{R}_{\tau\times\sigma} \in \mathbb{R}^{|\tau|\times|\sigma|}$, in order to avoid their re-calculation during each $\mathcal{H}$-matrix vector product.

In contrast, computing the entries in the Galerkin matrix for the boundary element method is very expensive. In this case, it is extremely important to further pre-compute and store the dense blocks $\boldsymbol{A}_{\tau\times\sigma}$. This has been realized in `hmglib`. The evaluation of the matrix entries is done in a batched way and the matrices are stored in GPU memory. To efficiently use the available memory, we store the system matrices continuously in GPU memory, without any padding. Pointers to the offsets of each matrix are passed to the batched matrix-vector product provided by `Magma`.

7.3.4 Distributed-memory parallelization

We aim at a distributed-memory, i.e. multi-GPU, parallelization for two reasons. First, we want to be able to solve large problems for which we need a high amount of (GPU) memory. However, GPUs are usually rather limited in terms of the available memory. This is why we need many GPUs (*scale-up*). Moreover, we want to be able to solve BEM problems as fast as possible (*speed-up*).

To fulfill both requirements, we first tried a matrix-based parallelization by dividing the large-scale system matrix into blocks of rows, on which we independently applied the $\mathcal{H}$-matrix approximation. However, this lead to a sub-optimal load balancing and sub-optimal speed-up, since large admissible matrix blocks were cut into smaller pieces. The approach further became prohibitive, as we ran into the situation that admissible blocks were divided into skinny (i.e. wide but thin) sub-blocks. Therefore, in the worst case, one of the low-rank factors in the ACA could become as large as the number of unknowns in the linear system times the required rank, with a strong negative impact on scale-up. By introducing a full (row- and column-wise) block-partitioning of the system matrix, we could remove this second issue. However, we did still cut large admissible blocks into smaller pieces.

Work load distribution parallelization. We overcome most of the mentioned issues by using a *work load distribution* parallelization instead of a matrix-based parallelization. In our work

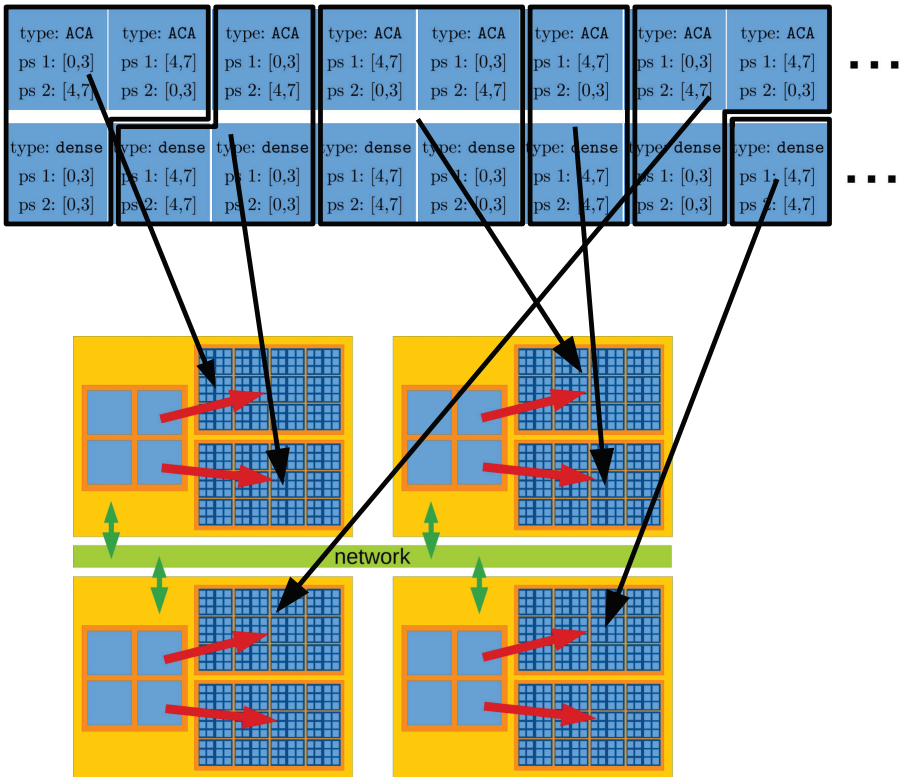


Figure 7.3: The distributed-memory parallelization of the $\mathcal{H}$-matrix approach distributes similar sized subsets of the dense and low-rank matrix blocks to the different GPUs. The yellow boxes represent hybrid compute nodes in an HPC system that are equipped with a four-core CPU and two GPUs.

load distribution parallelization, we focus specifically on problem size scale-up and calculation speed-up in the $\mathcal{H}$-matrix construction. This choice is valid, since the $\mathcal{H}$-matrix construction completely dominates the solution process in BEM applications.

Our work load distribution parallelization first builds on all GPUs the identical global block cluster tree and identifies admissible and non-admissible leafs. These leafs are put into two work load lists being identical on each GPU. Note that no system matrix entry has been evaluated at this stage. Then, each of the two work load lists is divided into p sub-lists, where p is the number of GPU processors. The very computationally expensive construction and storage of the low-rank or dense blocks in the sub-lists is done in a distributed way on each GPU. Figure 7.3 illustrates our parallelization concept. At the top part of this figure, we show the decomposed work load lists, which are then associated to one GPU. Our approach allows to fully decouple the construction of the $\mathcal{H}$-matrix. In practice, we use the Message Passing Interface (MPI) and associate one CPU process / thread to one GPU. The only parallelization information, which is required during the $\mathcal{H}$-matrix construction phase, is the (CPU-)process associated to each GPU. It is needed to distribute the sub-lists. The process number is provided by MPI.

As part of our current parallelization strategy, we store an identical copy of the vector involved in the $\mathcal{H}$-matrix-vector product on each GPU. The $\mathcal{H}$-matrix-vector product is applied individually on each GPU. That is, it is only performed for those admissible and non-admissible blocks that are available on the corresponding processor. To finalize the product, we use a global parallel reduction (summation) provided by MPI. For improved performance, we rely on a CUDA-aware MPI implementation [Kra13], such that we directly pass pointers to GPU memory to the MPI call. Data transfers to and from the network adapter are handled by MPI.

Up to this point, we did not discuss how to partition the work load lists into sub-lists. This has a strong impact on load balancing. In our current implementation, we stick to a rather simplistic scheme. It relies on the assumption that the amount of time required for a batched matrix-vector product of several dense matrices (or two matrix-vector products of skinny matrices in the ACA case) is proportional to the sum over the number of matrix entries of all matrices that are involved in the batched product. Concerning the dense matrix-vector product work load list, we thus balance the storage size of the batched matrices on each GPU. Similarly, we balance the storage size of the batched low-rank factors for the ACA work load list, compare Figure 7.3.

Scalability and load balancing discussion. Independently computing the block cluster tree on each GPU and further storing identical copies of the vector involved in the matrix-vector product requires, with growing problem size, a growing fixed amount of memory. This has a potential impact on the problem size scale-up on GPUs with a small amount of GPU memory. We could partially fix this obstruction by moving the block cluster tree construction to CPU. Nevertheless, our intention is to provide a solely GPU-based implementation. Therefore, we do not use the CPU.

Note, also that the currently required global communication in the $\mathcal{H}$-matrix-vector product might limit the speed-up in the matrix-vector product. We accept this, since we right-now focus on calculation speed-up in the $\mathcal{H}$-matrix construction, as discussed before. Finally, the underlying assumption for our work load list partitioning strategy strongly depends on the applied batched matrix-vector product implementation. Here, we see some room for improvements by a more elaborated cost model.

7.4 Numerical results

In the following, we will first briefly discuss the model problem and the applied GPU-based model BEM solver. This is followed by an overview of the used hard- and software and the definition of two test cases. The first major study of this section is concerned with numerical results that indicate convergence of the implemented method. The remaining part of this section discusses the performance and scalability of the multi-GPU approach.

7.4.1 Model BEM solver

To test our extended version of the `hmglib` library in the context of boundary element methods, we have implemented a GPU-based model BEM solver. It solves the model problem discussed in Example 7.1, i.e. the Laplace problem reformulated by the single-layer potential ansatz and

resulting in the boundary integral equation (7.3). We stress here that this BEM solver is solely built with the intention to have a test case for the `hmglib` library in the context of BEM. That is, it is not supposed to compete with other BEM libraries.

Our model GPU BEM solver is based on a sequential in-house code. This in-house code gets the boundary Γ by a parametric representation similar to iso-geometric analysis. It discretizes Γ by a quadrangular mesh and introduces a finite-dimensional trial space with piecewise constant ansatz functions. We identify basis functions with element centers. In the Galerkin matrix assembly, higher-order quadrature and the *Duffy trick* [Duf82, SS97] are applied to get an accurate approximation of the integrals. The resulting system of linear equations is solved by a conjugate gradient (CG) solver.

In our GPU version of the CPU code, we re-use the existing sequential CPU code to build the required data structures (mesh, element lists, ...). This data is copied to GPU. Then, the actual matrix assembly is done on GPU. To this end, we parallelize the fully decoupled node-wise assembly operation by appropriate CUDA device functions that overwrite the abstract matrix assembly class of `hmglib`. As e.g. discussed in [BC15], the complicated, memory-intensive and node-wise sequential quadrature routines easily lead to a limited GPU *utilization*. In fact, we had to limit the size of the so-called *thread blocks*, i.e. the number of threads executed on a symmetric multiprocessor of a GPU, to 128 in order to be able to run the quadrature routines. Typical choices for most other applications are 512 or 1024. The hand-written GPU-based CG solver uses the multi-GPU parallel $\mathcal{H}$-matrix-vector product. The solution of the iterative solver is copied back to CPU, where its error is evaluated by the standard error evaluation routines of the sequential CPU code.

All results of this work are calculated with the $\mathcal{H}$-matrix parameters $\eta = 1.0$ and $C_{leaf} = 32$. The stopping criterion of the CG solver is a relative residual of 10^{-8}.

7.4.2 Hardware and software setup

To run and benchmark `hmglib` together with the model GPU BEM solver, we use the former Top 1 HPC system *Titan* (27 Peta-FLOPS), located at the Oak Ridge National Lab, US. This is a Cray XK7 cluster equipped with 18688 compute nodes, which are connected by a Gemini interconnect. Each compute node contains a 16-core AMD Opteron processor, 32 GB of (CPU) memory and a Nvidia Tesla K20X GPU (Kepler architecture) with 6 GB of GPU memory.

We use Titan's default `gcc` compiler, Cray's `MPICH` implementation and Titan's (at time of benchmarking) latest `CUDA Toolkit` 7.0 (including the corresponding `Thrust` and `CUBLAS` libraries). In addition, we compile and use `Magma` 2.3.0 and `OpenBLAS` 0.2.20 (as dependency of `Magma`). In all compilations, we use the standard optimization flag `-O3`. All GPU codes are compiled for the best possible *Compute Architecture 3.5*. The version of `hmglib` that is used to create the results in this work is identical to the commit `8b4a4ff` of `hmglib` on Github [Zas18].

7.4.3 Test cases

To test our implementation, we first use the model geometry $\Omega := [0,1]^3$, i.e. Γ is the surface of the unit cube. As real-world test case, we further consider the geometry of a gearwheel with the bounding box $[-4,4] \times [-4,4] \times [-11.5, 9.1]$, as shown in Figure 7.4. In both cases, the

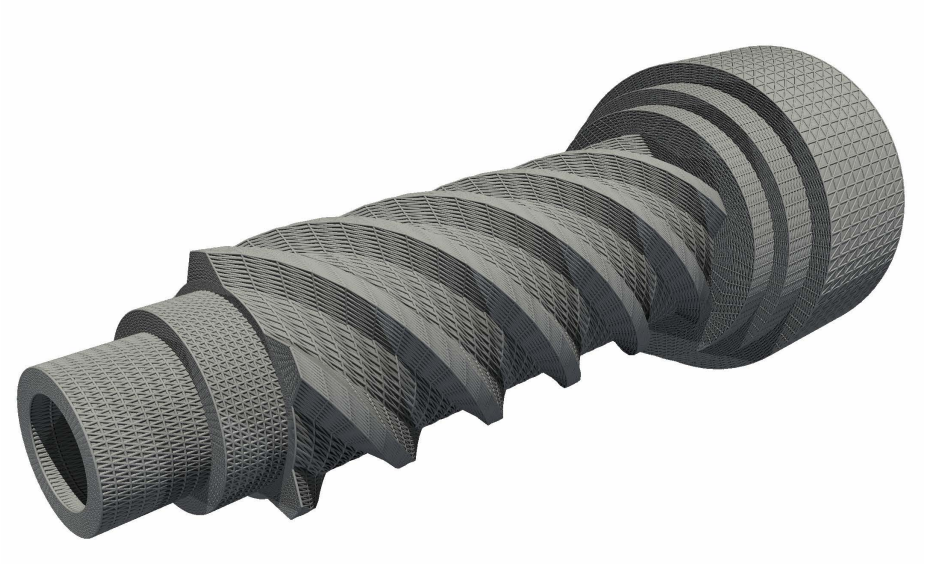


Figure 7.4: As real-world test case, we solve a boundary integral equation on a complex gearwheel geometry. The overlying mesh corresponds to a discretization with $N = 296960$ boundary elements.

right-hand side f in equation (7.3) is

$$f(\boldsymbol{x}) := 4{x_1}^2 - 3{x_2}^2 - {x_3}^2 \,.$$

Since the right-hand side is the trace of a harmonic function, this choice allows us to compare the numerical solution against the exact solution U_{exact} of the underlying Laplace equation. In particular, we compute the worst-case error

$$\epsilon(h) := \max_{\boldsymbol{x} \in X_{eval}} \left| U_{exact}(\boldsymbol{x}) - \tilde{\mathcal{S}} u_h(\boldsymbol{x}) \right|$$

by evaluating the single-layer potential ansatz from equation (7.2) for the approximated solution u_h. Here, X_{eval} is a large set of fixed points in the interior of the domain Ω.

7.4.4 Convergence

In order to verify the correctness of our GPU BEM model implementation and of the distributed-memory parallel $\mathcal{H}$-matrix implementation, we first do a classical convergence study, both with respect to the discretization, i.e. the number of boundary elements, and with respect to the approximation by the $\mathcal{H}$-matrix approach.

Cube geometry. We solve the above discussed model problem on the surface of the cube geometry for a growing number N of boundary elements. In this first experiment, we fix the approximation by the adaptive cross approximation to $k = 128$. The convergence study is done for $N = 1536, 6144, 24576, 98304, 393216$. Note that we compute the first two results on 4 GPUs and the remaining results on 128 GPUs. Although some of the problems would already

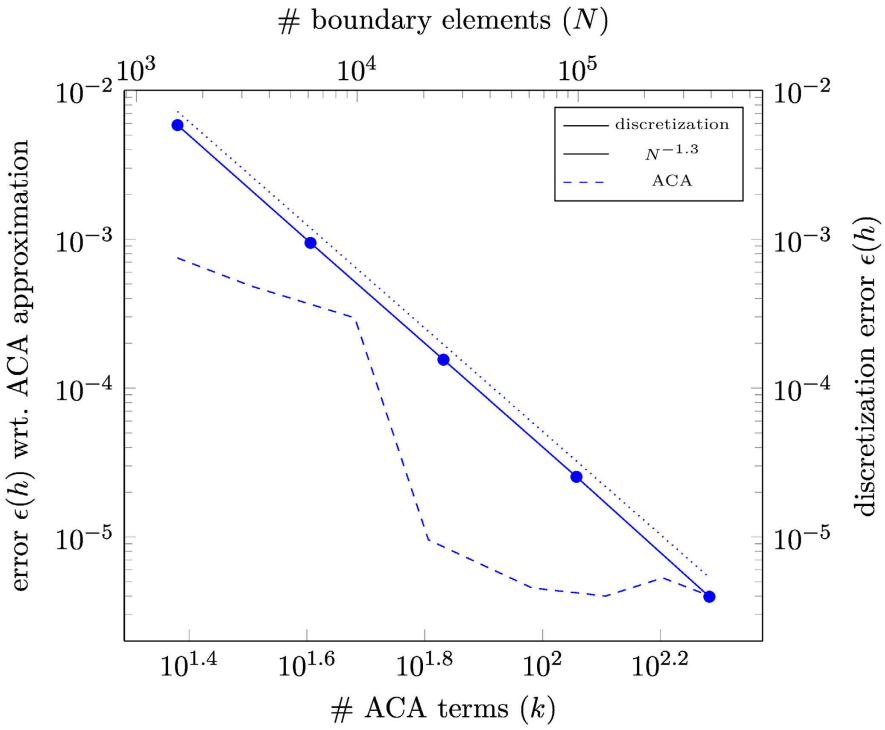


Figure 7.5: Convergence study for the multi-GPU implementation with model problem on a cube geometry. The discretization error *(solid line)* decays with an appropriate algebraic rate while the error in the ACA *(dashed line)* converges up to exponentially until it hits the discretization error.

fit in less GPUs (or even one GPU), we keep a higher number of GPUs to actually check the convergence of the *multi*-GPU code.

Figure 7.5 shows the convergence results for this first test by the solid blue line. The corresponding axes for this test are on the top and on the right-hand side. We observe an algebraic error decay with a (measured) rate of 1.3. If Γ would be smooth, we could get a rate of 1.5. However, since this is not the case, the observed rate perfectly fits our expectations.

We further check the convergence of the adaptive cross approximation. To this end, we fix the number of boundary elements to $N = 393216$ and gradually increase the number of terms used in the ACA as $k = 24, 32, 48, 64, 96, 128, 160, 192$. In this second test, we always use 128 GPUs. The results are depicted in Figure 7.5 as the dashed line. The error in the ACA decays up to exponentially until it hits the discretization error for roughly $k = 128$. Beyond that, it stagnates with small fluctuations.

Gearwheel geometry. Next, we repeat our previous studies with the complex real-world geometry of the gearwheel seen in Figure 7.4. We again fix the low-rank approximation to $k = 128$ and increase the number of boundary elements in accordance with $N = 18560, 74240, 296960, 1187840$. The first two problem sizes are computed on 256 GPUs and the second two problem sizes are computed on 1024 GPUs.

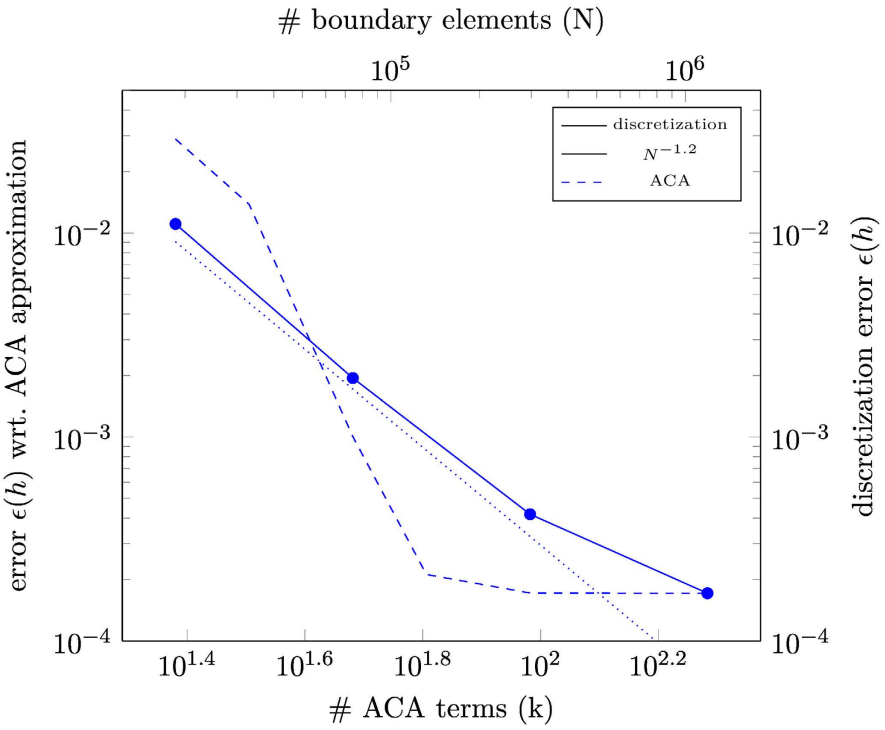


Figure 7.6: Our multi-GPU parallel BEM solver together with the `hmglib` library is also applied to the very complex gearwheel geometry. Here it shows a similar discretization error *(solid line)* and ACA approximation error *(dashed line)* as for the unit cube geometry.

The results are given in Figure 7.6 by the solid line. It shows a measured algebraic rate of roughly 1.2, which fits again our expectations since the gearwheel geometry is not smooth. For the largest problem size, we get a slight degradation in this rate. This might be due to our fixed stopping criterion of a relative residual of 10^{-8} in the CG solver and a potentially high condition number of the corresponding Galerkin system matrix.

We also repeat the convergence study for ACA for fixed $N = 1187840$ and growing $k = 24, 32, 48, 64, 96, 128, 160, 192$. As before, we observe an up to exponential convergence until the discretization error is hit. Beyond that, there are again only small variations on the same error level.

To summarize, our distributed-memory multi-GPU parallel model BEM code applied together with `hmglib` perfectly matches our convergence expectations. This holds for the model geometry of a unit cube and for the very complex gearwheel geometry.

7.4.5 Performance and scalability

To assess the properties of our implementation, we performed a series of benchmarking and scalability studies. All studies were carried out on *Titan*, cf. Section 7.4.2. Time measurements in our distributed-memory multi-GPU parallelization are wall clock times for the slowest parallel process, i.e. the slowest GPU. If not otherwise stated, these worst-case times were averaged

				runtime	
geometry	N	k	p	$\mathcal{H}$-setup [s]	CG solver [s/iter.]
cube	1536	24	1	0.86	0.0080
	6144	24	1	5.44	0.0147
	24576	24	1	39.91	0.0350
	98304	48	8	163.15	0.1504
	393216	48	32	698.49	0.0914
	1572864	48	128	1880.26	0.1918
	393216	24	128	46.66	0.0335
		32	128	101.01	0.0284
		48	128	245.35	0.0288
		64	128	342.47	0.0333
		128	128	518.79	0.0342
		160	128	555.77	0.0393
	1572864	48	128	1832.61	0.1124
		48	256	955.92	0.0858
		48	512	543.12	0.0767
		48	1024	338.80	0.0867
gearwheel	1187840	24	1024	497.41	0.0642
		32	1024	509.11	0.0585
		48	1024	534.10	0.0583
		64	1024	684.75	0.0600
		128	1024	864.29	0.0643
		160	1024	877.60	0.0783
	1187840	24	128	2726.23	0.0969
		24	256	1486.17	0.0827
		24	512	937.07	0.0633
		24	1024	497.90	0.0616

Table 7.1: The above table collects runtime benchmarks done with `hmglib` and our model GPU BEM solver on p GPUs. We are able to e.g. solve a BEM problem on a cube geometry with about 1.5 million boundary elements on GPUs in less than 6 minutes (5.7 minutes for the $\mathcal{H}$-matrix setup, 20 seconds for the CG solver).

over five runs to reduce the impact of changing loads on the utilized HPC system. Timings for all measurements are collected in Table 7.1.

Scale-up in problem size. We first discuss the scale-up in terms of problem size, exemplified for the cube geometry. The corresponding timings are given in the first block of Table 7.1. We are able to solve a problem with up to 24576 boundary elements and $k = 24$ on a single GPU. In this case, the setup time of the hierarchical matrix consumes about 40 seconds. A single iteration of the CG solver requires in average 0.035 seconds with a total of about 100 iterations.

To further increase the problem size, we increased the number of GPUs by a distributed-memory parallelization, cf. Table 7.1. Using our work load distribution parallelization, cf. Section 7.3.4, we are able to treat the cube geometry test case with about 1.5 million boundary elements on 128 GPUs for $k = 48$. The $\mathcal{H}$-matrix setup phase consumes about 31 minutes. Once the setup has been computed, the system of linear equations can be solved with only 0.19 seconds per iteration and a total runtime of about 75 seconds. That is, we do the linear solve with 1.5 million boundary elements in way less than one and a half minutes.

We also tried to further increase the problem size. However, we are not able to accomplish this since the amount of memory consumed on a single GPU (due to our strategy of independently carrying out the data structure setup on all GPUs) becomes too high for the 6 GB of GPU RAM of the Tesla K20X GPUs. The use of a more recent GPU like the Tesla P100 with 16 GB of GPU RAM would of course improve the situation. This GPU would also achieve a much higher performance in the $\mathcal{H}$-matrix setup, since our model BEM code would run much faster than on the rather old Tesla K20X cards. In the future, we also aim at combining a domain-decomposition parallelization with the work load distribution parallelization as for example in [Beb08, KMMZ18] to solve even much larger problem sizes.

Performance vs. accuracy. As discussed in Section 7.4.4, the increase in the number of terms utilized in the adaptive cross approximation has an important impact on the accuracy of the approximate solution. Therefore, we analyzed its impact on the performance for the cube geometry and the gearwheel geometry test case. In the first test case, we fix the problem size to about 400000 boundary elements and increase the number of terms k from 24 to up to 160. Note that the timings used in this paragraph correspond to a single test run instead of five test runs. Table 7.1 collects the runtime results of this test case in the second row block. From a theoretical point of view, we should expect a quadratic increase in the $\mathcal{H}$-matrix setup runtime with respect to k. In practice, this increase is visible in the $\mathcal{H}$-matrix setup (including the ACA approximation of the admissible blocks) for smaller k. However, for larger k, this increase becomes smaller. We assume that this behavior is due to the batching of the linear algebra operations involved in ACA. In fact, the larger the problem, the better it is possible to hide GPU latencies. We observe a much smaller runtime increase in the CG solver.

We tried the same experiment for the real-world gearwheel geometry test case with a problem size of about 1.2 million boundary elements with results given in the fourth row block of Table 7.1. Here, no quadratic runtime increase is visible. Instead, runtime is increased sub-linearly. In this test case, the runtime for the dense treatment of non-admissible matrix blocks seems to dominate the runtime. To summarize, an increase in the number of ACA terms has

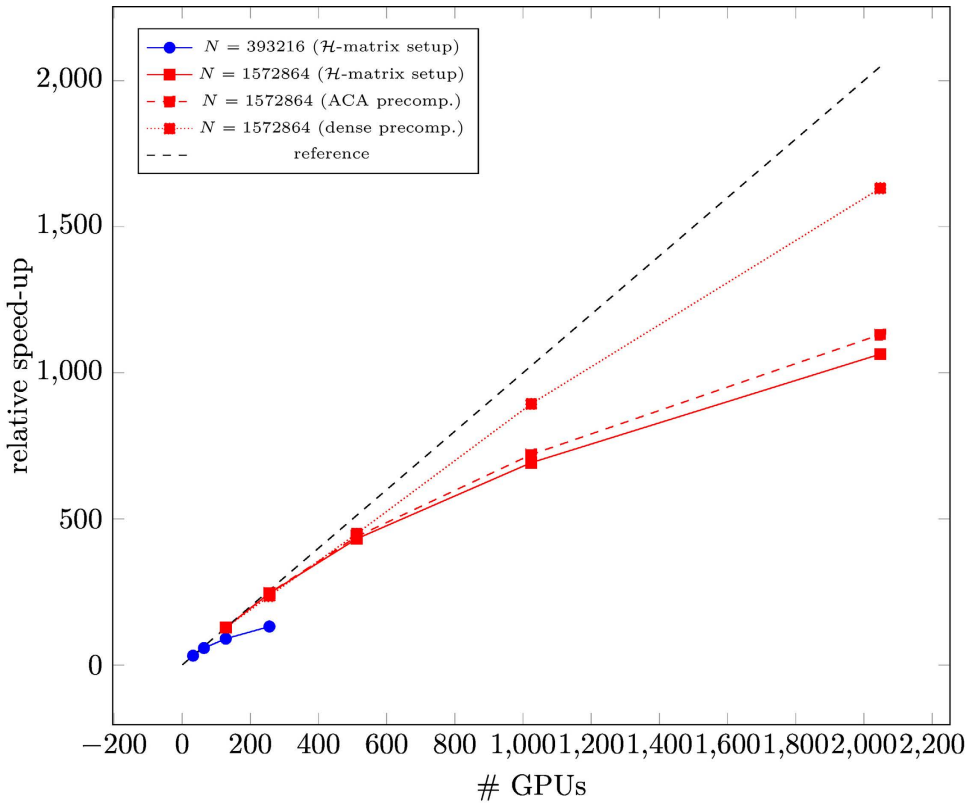


Figure 7.7: The parallel speed-up efficiency for $\mathcal{H}$-matrix setup of the cube geometry test case *(solid red line)* is above 67 percent on up to 1024 GPU (starting form 128 GPUs) and still reaches above 50 percent on 2048 GPUs. Focussing on the pre-computation of (dense) non-admissible blocks, we even achieve almost 80 percent parallel speed-up efficiency.

only a rather small impact on the overall runtime of our implementation.

Speed-up efficiency One of the main goals of this work is the increase in performance for the *$\mathcal{H}$-matrix setup* by the use of more GPUs. To showcase the achieved efficiency, we perform a parallel speed-up / strong scaling study for the cube and the gearwheel geometry. The results for the cube geometry are given in Figure 7.7 and in the third row block of Table 7.1.

In Figure 7.7, we show the results of a parallel speed-up analysis for problem sizes $N = 393216$ and $N = 1572864$ with $k = 48$. While the smaller problem size does not scale on large GPU counts, we observe a decent result for the larger test case. Here, a relative parallel speed-up efficiency of more than 67 percent is achieved starting from 128 GPUs and going to up to 1024 GPUs. The effective runtime of the $\mathcal{H}$-matrix setup on 1024 GPUs is less than 5.7 minutes, while the solution time by the CG solver is less than 20 seconds. In total, we therefore need on 1024 GPUs less than 6 minutes for the $\mathcal{H}$-matrix setup and solve for a problem size of more than 1.5 million boundary elements.

When trying to further speed-up the problem, we still achieve an acceptable speed-up ef-

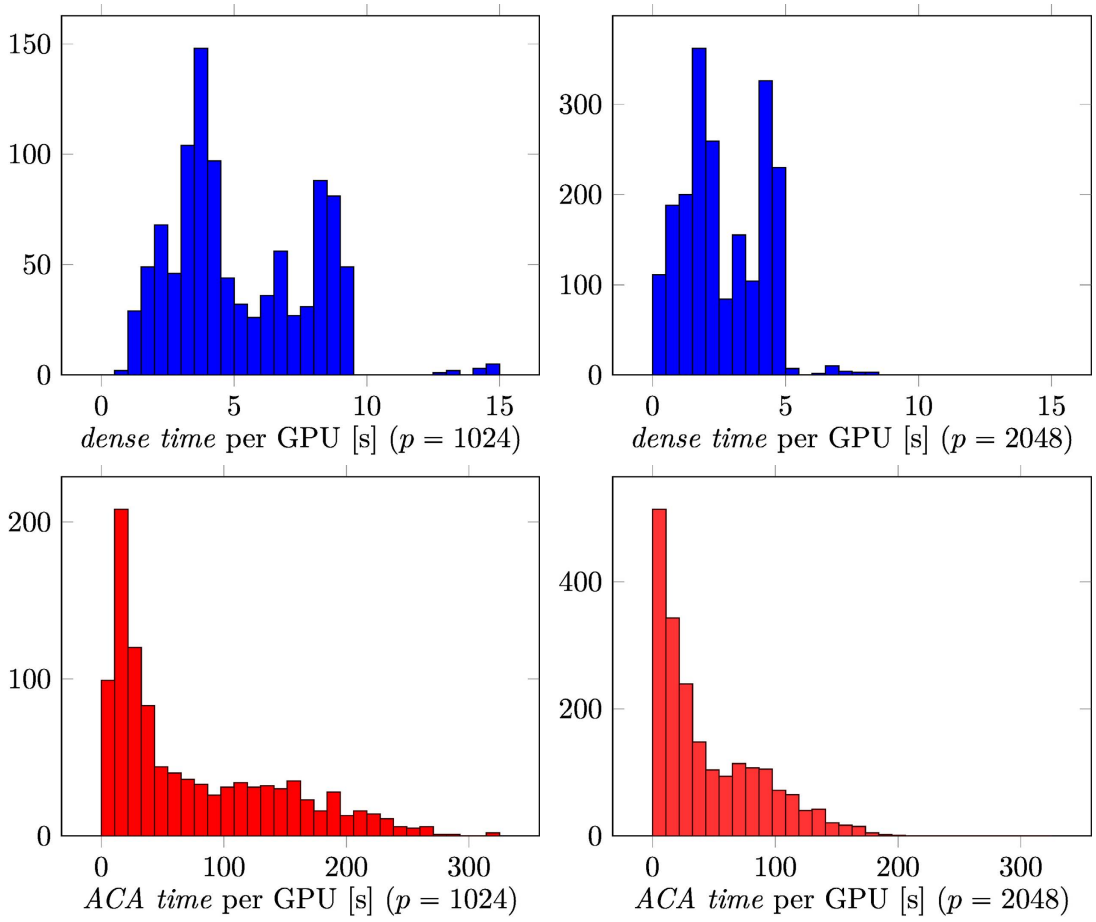


Figure 7.8: We analyze the load balancing in the speed-up study of the $\mathcal{H}$-matrix setup (cube geometry test case, $N = 1572864$, $k = 48$) with a special focus on the per-GPU runtimes of the pre-computation of the non-admissible blocks *(blue)* and the per-GPU runtimes of the approximation of the admissible blocks *(red)*.

ficiency of above 50 percent on 2048 GPUs. In addition, we looked more closely in the contribution to the scalability of the different work loads in the $\mathcal{H}$-matrix setup. As shown in Figure 7.7, the pre-computation of the (dense) non-admissible blocks scales much better than the approximation of the admissible blocks by ACA. From Table 7.1, we can depict that the strong scaling efficiency of the dense computations is in the range of 80 percent, even on 2048 GPUs. The scaling of the ACA work load is only a little bit more efficient than the overall $\mathcal{H}$-matrix setup phase. The overall setup phase is always less efficient than the dense and ACA work loads, since the GPU-parallel data structure setup, i.e. the block cluster tree traversal, is not parallelized in a distributed-memory manner, compare Section 7.3.4.

In Section 7.3.4, we also discussed the issue of load balancing. Effectively, we chose the model assumption that a roughly equal amount of batched matrix entries that are applied in a batched matrix-vector product will result in similar runtime performance. We check the quality of this assumption by examining the distribution of computational runtime over all

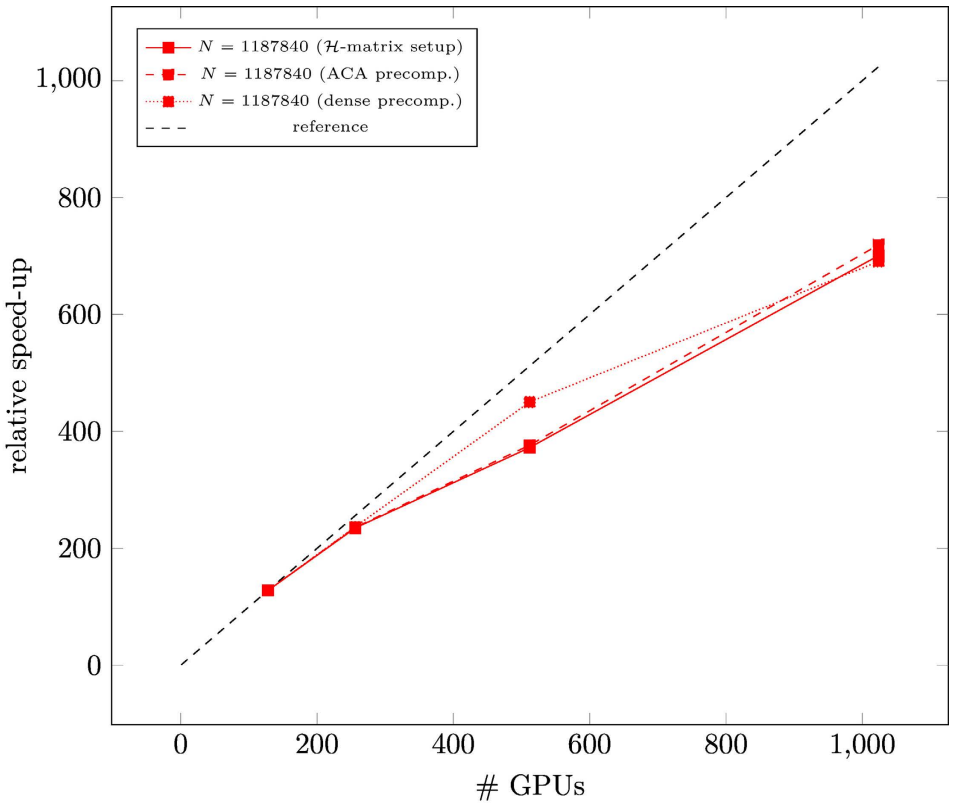


Figure 7.9: Considering the real-world test case of a gearwheel geometry with $N = 1187804$ and $k = 24$, we observe a relative parallel speed-up efficiency of above 68 percent going from 128 to 1024 GPUs. This result is almost identical to the much simpler cube geometry test case.

GPUs for the dense matrix blocks and the ACA matrix approximations. The results of this study for $N = 1572864$, $k = 48$ and $p = 1024, 2048$ are shown in Figure 7.8 as histogram plots. Qualitatively, the changes between 1024 and 2048 GPUs are only small, that is we only consider $p = 1024$. In practice, our model assumption does not yet lead to an optimal load balancing. While a major part of the timings for the dense matrix operations (see the blue histogram on the left-hand side in Figure 7.8) is nicely scattered around 5 seconds, we have some non-optimal outliers of up to 15 seconds. Nevertheless, the dense operations have only a moderate influence on the overall scalability due to their small maximum time. In contrast, the matrix block approximations by ACA (see the red histogram on the left-hand side in Figure 7.8) last up to more than 300 seconds. We especially observe the rather large portion of GPUs that only need a very small amount of time. In the future, we aim at improving the multi-GPU load balancing by techniques proposed e.g. in [Beb08, KMMZ18]. However, while these techniques work well in the context of non-batched operations, we assume that their combination with batching will still be sub-optimal on GPUs. Therefore, further research has to be carried out in order to improve the load balancing.

We finally repeat the parallel speed-up efficiency study for the gearwheel geometry test case

with $N = 1187840$ and $k = 24$. The speed-up results are graphically displayed in Figure 7.9. In this test case, the speed-up for the dense matrix work load and the ACA approximation work load are well aligned to the overall $\mathcal{H}$-matrix setup speed-ups. In total, we achieve above 68 percent of parallel speed-up efficiency going from 128 GPUs to 1024 GPUs. This is a decent result. Moreover, as shown by Table 7.1 in the last row block, the runtime for the $\mathcal{H}$-matrix setup on 1024 is about 8.3 minutes, while the per-iteration runtime of the CG solver is about 0.06 seconds with a runtime of less than 29 seconds for the full CG solve. That is, a total of 8.8 minutes is needed to solve a BEM problem with 1187840 boundary elements on a real-world geometry on 1024 (rather old) GPUs, recalling that the implemented GPU BEM solver is only intended to be a model application for the underlying `hmglib` library.

7.5 Conclusions

In this work, we considered the distributed-memory parallel multi-GPU parallel solution of boundary integral equations by the hierarchical matrix library `hmglib`. The main contribution of this work was the extension and distributed-memory parallelization of `hmglib`, such that Galerkin matrices from boundary element method discretizations given by arbitrary BEM codes can be approximated in a multi-GPU parallel way. Our multi-GPU parallelization of the $\mathcal{H}$-matrix library uses a work load distribution parallelization. Numerical studies and performance analysis were carried out with a model GPU BEM solver for piecewise constant boundary elements, which is based on an existing in-house CPU solver. This model GPU BEM solver was merely designed for a test of the `hmglib` library, however not with the intention to compete with other BEM solvers in the field. Our two numerical test cases showed, both, roughly 67 percent relative parallel speed-up efficiency going from 128 to 1024 GPUs on the GPU cluster *Titan*. This is a decent speed-up result. A cube geometry test case with about 1.5 million boundary elements could be solved within less than 6 minutes (5.7 minutes for the setup, 20 seconds for the CG solver) on 1024 GPUs. The real-world gearwheel geometry test case with about 1.2 million unknowns could be solved within 8.8 minutes on 1024 GPUs.

As future work, we consider the combination of a domain decomposition parallelization with our work load distribution parallelization. This new approach will introduce a distributed main data structure. Thereby it will be possible to scale to to much larger problem sizes. This will also hold for the CG solver. Moreover, we plan to introduce a further investigate load balancing techniques (e.g. as in the non $\mathcal{H}$-matrix work [DZSS17]). An especially apealing approach could be the introduction of a model for the runtimes of the batched BLAS operations an the matrix assembly quadrature routines.

Acknowledgements

This work is funded by the Swiss National Science Foundation (SNF) under project number 407540_167186. Furthermore, code development and benchmarking tasks in this research were done on resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. All funding and support is gratefully acknowledged.

References

[ALM+18] K. Akbudak, H. Ltaief, A. Mikhalev, A. Charara, and D. E. Keyes. Exploiting data sparsity for large-scale matrix computations. Technical report, King Abdullah University of Science and Technology, 2018.

[BÏ7] S. Börm. `H2Lib`, a library for hierarchical matrices. Online, 2017. http://www.h2lib.org (last access: 2018/06/18).

[BC15] S. Börm and S. Christophersen. Approximation of BEM matrices using gpgpus. *CoRR*, abs/1510.07244, 2015.

[Beb] M. Bebendorf. `AHMED` Another software library on hierarchical matrices for elliptic differential equations. Online. https://github.com/xantares/ahmed (last access: 2018/06/18).

[Beb08] M. Bebendorf. *Hierarchical Matrices. A Means to Efficiently Solve Elliptic Boundary Value Problems*, volume 63 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin, 2008.

[BGH03] S. Börm, L. Grasedyck, and W. Hackbusch. Introduction to hierarchical matrices with applications. *Engineering analysis with boundary elements*, 27(5):405–422, 2003.

[BK09] M. Bebendorf and S. Kunis. Recompression techniques for adaptive cross approximation. *Journal of Integral Equations and Applications*, 21(3):331–357, 2009.

[Bör04] S. Börm. $\mathcal{H}^2$-matrices. Multilevel methods for the approximation of integral operators. *Computing and Visualization in Science*, 7(3):173–181, Oct 2004.

[BR03] M. Bebendorf and S. Rjasanow. Adaptive low-rank approximation of collocation matrices. *Computing*, 70(1):1–24, 2003.

[BTLK18] W. H. Boukaram, G. Turkiyyah, H. Ltaief, and D. E. Keyes. Batched QR and SVD algorithms on GPUs with applications in hierarchical matrix compression. *Parallel Computing*, 74:19–33, 2018. Parallel Matrix Algorithms and Applications (PMAA'16).

[CKL17] A. Charara, D. E. Keyes, and H. Ltaief. Batched triangular dense linear algebra kernels for very small matrix sizes on gpus. Technical report, King Abdullah University of Science and Technology, 2017.

[CKL18] A. Charara, D. E. Keyes, and H. Ltaief. Batched tile low-rank GEMM on GPUs. Technical report, King Abdullah University of Science and Technology, 2018.

[DGH+14] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki. Accelerating numerical dense linear algebra calculations with GPUs. *Numerical Computations with GPUs*, pages 1–26, 2014.

[DHK+18] J. Dölz, H. Harbrecht, S. Kurz, S. Schöps, and F. Wolf. A fast isogeometric BEM for the three dimensional Laplace- and Helmholtz problems. *Computer Methods in Applied Mechanics and Engineering*, 330:83–101, 2018.

[Duf82] M. G. Duffy. Quadrature over a pyramid or cube of integrands with a singularity at a vertex. *SIAM Journal on Numerical Analysis*, 19(6):1260–1262, 1982.

[DZSS17] A. Derler, R. Zayer, H. Seidel, and M. Steinberger. Dynamic scheduling for efficient hierarchical sparse matrix operations on the GPU. In W. D. Gropp, P. Beckman, Z. Li, and F. J. Cazorla, editors, *Proceedings of the International Conference on Supercomputing, ICS 2017, Chicago, IL, USA, June 14-16, 2017*, pages 7:1–7:10. ACM, 2017.

[GKLB08] L. Grasedyck, R. Kriemann, and S. Le Borne. Parallel black box-LU preconditioning for elliptic boundary value problems. *Computing and Visualization in Science*, 11(4):273–291, 2008.

[GKW03] L. Gaul, M. Kögler, and M. Wagner. *Boundary Element Methods for Engineers and Scientists*. Springer, Berlin, 2003.

[GLR+16] P. Ghysels, X. S. Li, F. Rouet, S. Williams, and A. Napov. An efficient multicore implementation of a novel HSS-structured multifrontal solver using randomized sampling. *SIAM Journal on Scientific Computing*, 38(5):358–384, 2016.

[GR97] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numerica*, 6:229–269, 1997.

[Hac15] W. Hackbusch. *Hierarchical matrices: Algorithms and Analysis*, volume 49 of *Springer series in computational mathematics*. Springer, Berlin, 2015.

[Hac16] W. Hackbusch. Survey on the technique of hierarchical matrices. *Vietnam Journal of Mathematics*, 44(1):71–101, 2016.

[Hal08] T. R. Halfhill. Parallel Processing with CUDA. *Microprocessor Report*, January 2008.

[HB02] W. Hackbusch and S. Börm. $\mathcal{H}^2$-matrix approximation of integral operators by interpolation. *Applied Numerical Mathematics*, 43(1-2):129–143, 2002.

[HB10] J. Hoberock and N. Bell. Thrust: A parallel template library, 2010. Version 1.7.0.

[HDT+15] A. Haidar, T. Dong, S. Tomov, P. Luszczek, and J. Dongarra. Framework for batched and GPU-resident factorization algorithms to block Householder transformations. In *ISC High Performance 2015*, pages 31–47, Cham, 2015. Springer International Publishing.

[HKS00] W. Hackbusch, B. Khoromskij, and S. A. Sauter. On $\mathcal{H}^2$-matrices. In *Lectures on Applied Mathematics: Proceedings of the Symposium Organized by the Sonderforschungsbereich 438 on the Occasion of Karl-Heinz Hoffmanns 60th Birthday, Munich, June 30–July 1, 1999*, pages 9–29, Berlin-Heidelberg, 2000. Springer.

[HN89] W. Hackbusch and Z. P. Nowak. On the fast matrix multiplication in the boundary element method by panel clustering. *Numerische Mathematik*, 54(4):463–491, 1989.

[HR10] H. Harbrecht and M. Randrianarivony. From computer aided design to wavelet BEM. *Computing and Visualization in Science*, 13(2):69–82, 2010.

[HW08] G. C. Hsiao and W. L. Wendland. *Boundary Integral Equations*, volume 164 of *Applied Mathematical Sciences*. Springer, Berlin, 2008.

[KMMZ18] M. Kravcenko, L. Maly, M. Merta, and J. Zapletal. Parallel assembly of ACA BEM matrices on Xeon Phi clusters. In R. Wyrzykowski, J. Dongarra, E. Deelman, and K. Karczewski, editors, *Parallel Processing and Applied Mathematics*, pages 101–110, Cham, 2018. Springer International Publishing.

[Kra13] J. Kraus. An introduction to CUDA-aware MPI. Online, 2013. NVIDIA Developer Blog.

[Kri05] R. Kriemann. Parallel $\mathcal{H}$-matrix arithmetics on shared memory systems. *Computing*, 74(3):273–297, 2005.

[Kri13] R. Kriemann. $\mathcal{H}$-LU factorization on many-core systems. *Computing and Visualization in Science*, 16(3):105–117, 2013.

[Kri17] R. Kriemann. $\mathcal{H}$-Lib$^{\text{pro}}$. Online, 2017. http://www.hlibpro.com (last access: 2018/06/18).

[LGS+09] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH construction on GPUs. *Computer Graphics Forum*, 28(2):375–384, 2009.

[Mor66] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical Report Ottawa, Ontario, Canada, 1966.

[MZ18] M. Merta and J. Zapletal. A parallel library for boundary element discretization of engineering problems. *Mathematics and Computers in Simulation*, 145:106–113, 2018.

[MZBF15] B. Marussig, J. Zechner, G. Beer, and T.-P. Fries. Fast isogeometric boundary element method based on independent field approximation. *Computer Methods in Applied Mechanics and Engineering*, 284:458–488, 2015.

[OYIY18] S. Ohshima, I. Yamazaki, A. Ida, and R. Yokota. Optimization of hierarchical matrix computation on GPU. In R. Yokota and W. Wu, editors, *Supercomputing Frontiers*, pages 274–292, Cham, 2018. Springer International Publishing.

[Pou] J. Poulson. `DMHM` - Distributed-Memory Hierarchical Matrices. Online. https://bitbucket.org/poulson/dmhm (last access: 2018/06/18).

[RLGN16] F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov. A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization. *ACM Transactions on Mathematical Software*, 42(4):27:1–35, 2016.

[SBA+15] W. Śmigaj, T. Betcke, S. Arridge, J. Phillips, and M. Schweiger. Solving boundary integral problems with BEM++. *ACM Transactions on Mathematical Software*, 41(2):6:1–40, 2015.

[SDC07] Z. Sheng, P. Dewilde, and S. Chandrasekaran. Algorithms to solve hierarchically semi-separable systems. In D. Alpay and V. Vinnikov, editors, *System Theory, the Schur Algorithm and Multidimensional Analysis*, pages 255–294, Basel, 2007. Birkhäuser.

[SS97] S. A. Sauter and C. Schwab. Quadrature for *hp*-Galerkin BEM in $\mathbb{R}^3$. *Numerische Mathematik*, 78(2):211–258, 1997.

[SS11] S. A. Sauter and C. Schwab. *Boundary Element Methods*. Springer Series in Computational Mathematics. Springer, Berlin–Heidelberg, 2011.

[Ste08] O. Steinbach. *Numerical Approximation Methods for Elliptic Boundary Value Problems*. Springer, New York, 2008.

[VBD17] K. Vater, T. Betcke, and B. Dilba. Simple and efficient GPU parallelization of existing $\mathcal{H}$-matrix accelerated BEM code. *CoRR*, abs/1711.01897, 2017.

[YAI+18] I. Yamazaki, A. Abdelfattah, A. Ida, S. Ohshima, S. Tomov, R. Yokota, and J. Dongarra. Performance of hierarchical-matrix bicgstab solver on GPU clusters. In *Proc. IEEE Int. Parallel and Distributed Processing Symp. (IPDPS)*, pages 930–939, May 2018.

[Zas17] P. Zaspel. Algorithmic patterns for $\mathcal{H}$-matrices on many-core processors. *CoRR*, abs/1708.09707, 2017.

[Zas18] P. Zaspel. `hmglib` - Simple H matrix library on GPU. Online, 2018. https://github.com/zaspel/hmglib (last access: 2018/06/18).