Application of Hierarchical Matrices For Solving Multiscale Problems

Von der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

DISSERTATION

zur Erlangung des akademischen Grades
DOCTOR RERUM NATURALIUM
(Dr. rer. nat.)

im Fachgebiet

Numerische Mathematik

vorgelegt

von Diplommathematiker Alexander Litvinenko

geboren am 31.08.1979 in Almaty, Kasachstan

Die Annahme der Dissertation haben empfohlen:

1. Prof. Dr. Dr. h.c. Wolfgang Hackbusch (MPIMN Leipzig)
2. Prof. Dr. Ivan G. Graham (University Bath, UK)
3. Prof. Dr. Sergey Rjasanov (Universität des Saarlandes)

Die Verleihung des akademischen Grades erfolgt auf Beschluss
des Rates der Fakultät für Mathematik und Informatik
vom **20.11.2006** mit dem Gesamtprädikat **cum laude**.

Acknowledgement

I would like to thank the following people:

- Prof. Dr. Dr. h.c. Wolfgang Hackbusch for inviting me to the Max Planck Institute for Mathematics in the Sciences in Leipzig, for the very modern theme, for his ideas and for obtaining financial support. I am thankful for his enjoyable lecture courses: "Elliptic Differential Equations", "Hierarchical Matrices" and "Iterative Solution of Large Sparse Systems of Equations". These courses were very useful during my work.

- PD DrSci. Boris N. Khoromskij (PhD) for his supervisory help at all stages of my work, for useful corrections and fruitful collaboration in certain applications, and also for his inspiring lecture courses "Data-Sparse Approximation of Integral Operators" and "Introduction to Structured Tensor-Product Representation".

- Dr. Lars Grasedyck and Dr. Steffen Börm for their patience in explaining the $\mathcal{H}$-matrix technique and details of HLIB,

- Dr. Ronald Kriemann for his advice in programming,

- Prof. Dr. Dr. h.c. Wolfgang Hackbusch, Prof. Dr. Ivan G. Graham (University of Bath, England) and Prof. Dr. Sergey Rjasanov (Universität des Saarlandes, Germany) for agreeing to referee this thesis.

This PhD work was done in the Max Planck Institute for Mathematics in the Sciences. I am deeply appreciative of the help of Mrs. Herrmann, Mrs. Hünniger and Mrs. Rackwitz from the personnel department of the institute. I am equally grateful to the DFG fond for the program "Analysis, Modelling and Simulation of Multiscale problems".

I would like to thank all my colleagues and friends Mike Espig, Isabelle Greff, Lehel Banjai, Petya Staykova, Alexander Staykov, Michail Perepelitsa, Graham Smith and all the others for making my time in Leipzig so enjoyable.

Particular thanks go to Henriette van Iperen for her moral support and the countless hours spent correcting the English version of this text.

And last, but certainly not least I would like to thank my wife for her unfailing, loving support during all these years.

Contents

Notation

Ω, ω	polygonal domains
$\partial\Omega$, $\partial\omega$	external boundaries of Ω and ω
Γ	a part of the external boundary $\partial\omega$
γ_ω, γ	interface in ω
$C_0^\infty(\Omega)$	infinitely differentiable functions with compact supports
f	right-hand side
h, H	mesh sizes
$H^k(\Omega), H_0^k(\Omega)$	Sobolev spaces
I, J	index sets, e.g., $I = \{0, 1, 2, ..., n-1\}$
L	differential operator
A	stiffness matrix
$_h$	grid step size
L_h, A	matrix of a finite system of equations
L^∞	space of essentially bounded functions
L^2	space of square-integrable functions
$\mathcal{O}(\cdot)$	Landau symbol: $f(x) = \mathcal{O}(g(x))$ if $\|f(x)\| \leq$ const $\|g(x)\|$
$\mathbb{R}$, $\mathbb{R}_+$	real numbers, positive real numbers
supp f	support of the function f
u	analytic solution
u_h	discrete solution
$\mathbf{c}$	discrete right-hand side
V_h	finite-element space
$\partial\Omega$, $\partial\omega$	external boundaries of the domains Ω and ω
Δ	the Laplace operator
$(\cdot,\cdot)_{L^2(\Omega)}$	scalar product on $L^2(\Omega)$
$\|\cdot\|_{L^2(\Omega)}$	norm on $L^2(\Omega)$
$\|\cdot\|_2$	Euclidean norm or spectral norm
$\|\cdot\|_\infty$	maximum norm
T_I, T_J	cluster trees
$T_{I\times J}$	block cluster tree
$\mathcal{H}$, $\mathcal{H}(T_{I\times J}, k)$	class of hierarchical matrices with a maximal
$A^{-\mathcal{H}}$	$\mathcal{H}$-matrix approximant to the inverse of A
	low-rank k and with a block cluster tree $T_{I\times J}$
$R(k, n, m)$	class of low-rank matrices with n rows, m columns and with a rank k
$\oplus, \ominus, \odot$	formatted arithmetic operations in the class of hierarchical matrices
$\oplus_k, \ominus_k, \odot_k$	formatted arithmetic with the fixed rank k
$P_{h\leftarrow H}$	prolongation matrix
$\alpha(x)$	coefficients in a differential equation, e.g. jumping or oscillating ones
$\mathcal{F}_h$, $\mathcal{F}_H$	discrete solution operators, applied to the right-hand side.
$\mathcal{G}_H$	discrete solution operator, applied to the Dirichlet data
d	spatial dimension, e.g. $\mathbb{R}^d$, $d = 1, 2, 3$
ν	frequency, e.g. $sin(\nu x)$

$\mathbf{x}$, $\mathbf{y}$	nodal points in Ω, e.g. $\mathbf{x} = (x_1, ..., x_d)$
$\mathbf{u}$	solution vector $\mathbf{u} = (u_1, ..., u_N)^T$
log	natural logarithm based 2
dof	degree of freedom
$n_h(\omega_i)$	number of nodal points in a domain ω_i with the grid step size h
$n_{h,x}$,$n_{h,y}$	number of nodal points in ox and oy directions
q	number of processors
$cond(A)$	condition number of a matrix A
$\lambda_{max}(A)$,$\lambda_{min}(A)$	maximal and minimal eigenvalues of a matrix A
Ψ^g, Ψ^g_ω	boundary-to-boundary mapping
Ψ^f, Ψ^f_ω	domain-to-boundary mapping
Φ^g, Φ^g_ω	boundary-to-interface mapping
Φ^f, Φ^f_ω	domain-to-interface mapping
$\mathcal{T}_h$, $\mathcal{T}_H$	triangulations with the grid step sizes h and H
$T_{\mathcal{T}_h}$, $T_{\mathcal{T}_H}$	domain decomposition trees with the triangulations $\mathcal{T}_h$, $\mathcal{T}_H$
$T^{\geq H}_{\mathcal{T}_h}$, $T^{\leq H}_{\mathcal{T}_h}$	two parts of the domain decomposition tree $T_{\mathcal{T}_h}$
N_f, N_g	computational complexities of Φ^f and Φ^g
$S(\Phi)$	storage requirement for a mapping Φ
$global\_k$	maximal rank of the non-diagonal admissible subblocks in an $\mathcal{H}$-matrix
$\tilde{\mathbf{u}}_k$	solution obtained by the HDD method; the subindex k indicates that the fixed rank arithmetic is used (see Def. 5.9.3)
$\tilde{\mathbf{u}}_\varepsilon$	solution obtained by the HDD method; the subindex ε indicates that the adaptive rank arithmetic is used (see Def. 5.9.3)
$\tilde{\mathbf{u}}_L$	solution of $A\mathbf{u} = \mathbf{c}$, $A = LL^T$, $\mathbf{u}_L = (L^T)^{-\mathcal{H}} L^{-\mathcal{H}} \mathbf{c}$
ε_{cg}	the value which is used for the stopping criterium in CG
$\tilde{\mathbf{u}}_{cg}$	solution obtained by the PCG method (with $\mathcal{H}$-Cholesky preconditioner)
ε_a	parameter for the adaptive rank arithmetic
ε_h	discretisation error
$\varepsilon_{\mathcal{H}}$	$\mathcal{H}$-matrix approximation error
n_{min}	minimal size of an inadmissible block (see Section 5.5.2), by default, $n_{min} = 32$
$I(\omega_h)$	index set of nodal points in ω
$I(\partial\omega_h)$	index set of nodal points on $\partial\omega$
$I(\gamma)$, $I(\gamma_\omega)$	index set of nodal points on γ and γ_ω respectively
HMM	Hierarchical Multiscale Method
HDD	Hierarchical Domain Decomposition method
CG	conjugate gradient
PCG	preconditioned conjugate gradient.

1 Introduction

Zu neuen Ufern lockt ein neuer Tag,
J.W. von Goethe

In this work we combine hierarchical matrix techniques and domain decomposition methods to obtain fast and efficient algorithms for the solution of multiscale problems. This combination results in the hierarchical domain decomposition method (HDD).

- *Multiscale problems* are problems that require the use of different length scales. Using only the finest scale is very expensive, if not impossible, in computer time and memory.
- A *hierarchical matrix* $M \in \mathbb{R}^{n\times m}$ (which we refer to as an $\mathcal{H}$-matrix) is a matrix which consists mostly of low-rank subblocks with a maximal rank k, where $k \ll \min\{n, m\}$. Such matrices require only $\mathcal{O}(kn \log n)$ (w.l.o.g. $n \geq m$) units of memory. The complexity of all arithmetic operations with $\mathcal{H}$-matrices is $\mathcal{O}(k^\alpha n \log^\alpha n)$, where $\alpha = 1, 2, 3$. The accuracy of the $\mathcal{H}$-matrix approximation depends on the rank k.
- *Domain decomposition methods* decompose the complete problem into smaller systems of equations corresponding to boundary value problems in subdomains. Then fast solvers can be applied to each subdomain. Subproblems in subdomains are independent, much smaller and require less computational resources as the initial problem.

The model problem we shall consider in this thesis is the elliptic boundary value problem with L^∞ coefficients and with Dirichlet boundary condition:

$$\begin{cases} Lu = f & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega, \end{cases} \tag{1.1}$$

whose coefficients may contain a non-smooth parameter, e.g.,

$$L = -\sum_{i,j=1}^{2} \frac{\partial}{\partial_j} \alpha_{ij} \frac{\partial}{\partial_i} \tag{1.2}$$

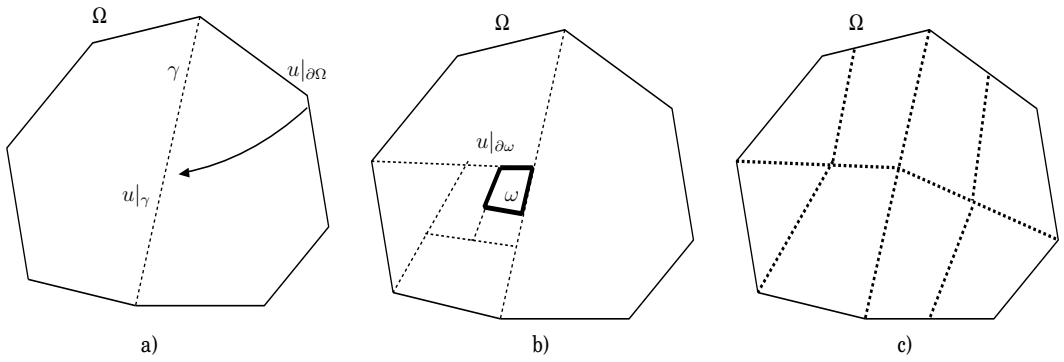
with $\alpha_{ij} = \alpha_{ji}(\mathbf{x}) \in L^\infty(\Omega)$ such that the matrix function $\mathcal{A}(\mathbf{x}) = (\alpha_{ij})_{i,j=1,2}$ satisfies $0 < \underline{\lambda} \leq \lambda_{\min}(\mathcal{A}(\mathbf{x})) \leq \lambda_{\max}(\mathcal{A}(\mathbf{x})) \leq \overline{\lambda}$ for all $\mathbf{x} \in \Omega \subset \mathbb{R}^2$. This setting allows us to treat oscillatory as well as jumping coefficients.

This equation can represent incompressible single-phase porous media flow or steady state heat conduction through a composite material. In the single-phase flow, u is the flow potential and α is the permeability of the porous medium. For heat conduction in composite materials, u is the temperature, $q = -\alpha\nabla u$ is the heat flow density and α is the thermal conductivity.

Examples of the typical problems

Suppose the solution on the boundary $\partial\Omega$ (Fig. 1 (a)) is given. Denote the solution on the interface γ by $u|_\gamma$.

In the domain decomposition society a fast and efficient procedure for computing



the solution $u|_\gamma$, which depends on both the right-hand side and the boundary data $u|_{\partial\Omega}$ is of interest. In another problem setup (Fig. 1 (b)) the solution in a small subdomain $\omega \subset \Omega$ is of interest. For example, the initial domain is an airplane and for constructive purposes the solution (or flux) in the middle of both wings is of interest. At the same time, to compute the solution in the whole airplane is very expensive. To solve the problem in ω the boundary values on $\partial\omega$ are required. How do we produce them efficiently from the global boundary data $u|_{\partial\Omega}$ and the given right-hand side f? To solve the initial problem in parallel (e.g., on a machine with eight processors) the solution on the interface (see Fig. 1 (c)) is required. How do we compute the solution on this interface effectively? The last problem setup is also required for multiscale problems. E.g., the interface in Fig. 1 (c) may be considered as a coarse grid. In multiscale problems often only the solution on a coarse grid is of interest. The subdomains can be considered as "cells" with periodic structures. In this work we explain how the offered method (HDD) can be applied for solving such problems.

Review of classical methods

After an FEM discretisation of (1.1), we obtain the system of linear equations

$$A\mathbf{u} = \mathbf{c}, \tag{1.3}$$

where the stiffness matrix A is large and sparse (e.g., $A \in \mathbb{R}^{10^6 \times 10^6}$ for the Laplace operator). There exist different methods for solving this system, for example, direct methods (Gauss elimination, method of $\mathcal{H}$-matrices, direct domain decomposition), iterative methods (multigrid, Conjugate Gradients), and combinations of the previous methods (CG with the hierarchical LU factorization as a preconditioner).

The direct methods (Gauss, LU) do not have convergence problems, but they require a computational cost of $\mathcal{O}(n^3)$, where n is the number of unknowns. For the same reason, they are insufficient if the coefficients of the operator L belong to different scales. Iterative methods produce approximations $\mathbf{u}^n$ converging to the exact solution $\mathbf{u}^*$, but do not compute the matrix A^{-1}. Multigrid methods compute the solution on the coarsest grid and then extend the solution from the coarse to a fine grid. The multigrid iterations use a smoothing procedure to decrease the approximation error from one grid to another.

The $\mathcal{H}$-matrix method takes into account the structure and properties of the continuous operator and builds a special block matrix where almost all blocks are approximated by low-rank matrices. The method of $\mathcal{H}$-matrices was developed by Hackbusch and others [33]. Papers [9], [46] have shown that $\mathcal{H}$-matrices can be used as preconditioners (e.g., the hierarchical LU factorisation, denoted by $\mathcal{H}$-LU). The preconditioners based on the $\mathcal{H}$-matrices are fast to compute (the cost being $\mathcal{O}(n \log^2 n)$). As the accuracy of the $\mathcal{H}$-matrix approximation increases fewer iteration steps are required. The $\mathcal{H}$-matrix approximation with high accuracy can be used as a direct solver.

Domain decomposition methods together with $\mathcal{H}$-matrix techniques were applied in [35], [38].

A brief description of the HDD method

The idea of the HDD method belongs to Hackbusch ([34]). Let k be the maximal rank which is used for $\mathcal{H}$-matrices (see Chapter 5), n_h and n_H the numbers of degrees of freedom on a fine grid and on a coarse grid, respectively. In order to better understand the HDD method, we now list its properties:

1. The complexities of the one-grid version and two-grid version of HDD are

$$\mathcal{O}(k^2 n_h \log^3 n_h) \quad \text{and} \quad \mathcal{O}(k^2 \sqrt{n_h n_H} \log^3 \sqrt{n_h n_H})$$

   respectively.

2. The storage requirements of the one-grid version and two-grid version of HDD are

$$\mathcal{O}(k n_h \log^2 n_h) \quad \text{and} \quad \mathcal{O}(k \sqrt{n_h n_H} \log^2 \sqrt{n_h n_H})$$

   respectively.

3. HDD computes two discrete hierarchical solution operators $\mathcal{F}_h$ and $\mathcal{G}_h$ such that:

$$u_h = \mathcal{F}_h f_h + \mathcal{G}_h g_h, \tag{1.4}$$

where $u_h(f_h, g_h)$ is the FE solution of (1.1), f_h the FE right-hand side, and g_h the FE Dirichlet boundary data. Both operators $\mathcal{F}_h$ and $\mathcal{G}_h$ are approximated by $\mathcal{H}$-matrices.

4. HDD allows one to compute two operators $\mathcal{F}_H$ and $\mathcal{G}_h$ such that:

$$u_h = \mathcal{F}_H f_H + \mathcal{G}_h g_h, \tag{1.5}$$

where $F_H := \mathcal{F}_h P_{h \leftarrow H}$, $P_{h \leftarrow H}$ is the prolongation matrix (see Section 4.3.6), f_H the FE right-hand side defined on a coarse scale with step size H and $\mathcal{F}_H$ requires much less computational resources as $\mathcal{F}_h$.

5. A very low cost computation of different functionals of the solution is possible, for example:
   a) Neumann data $\frac{\partial u_h}{\partial n}$ at the boundary,
   b) mean values $\int_\omega u_h d\mathbf{x}$, $\omega \subset \Omega$, solution at a point or solution in a small subdomain ω,
   c) flux $\int_C \nabla u \overrightarrow{n} d\mathbf{x}$, where C is a curve in Ω.

6. It provides the possibility of finding u_h restricted to a coarser grid with reduced computational resources.

7. Because of (1.4), HDD shows big advantages in complexity for problems with multiple right-hand sides and multiple Dirichlet data.

8. HDD is easily parallelizable.

9. If the initial problem contains repeated patterns then the computational resources can be drastically reduced.

In particular, the HDD method is well-suited for solving multiscale problems.

The diagram in Fig. 1.1 represents the content of this work.

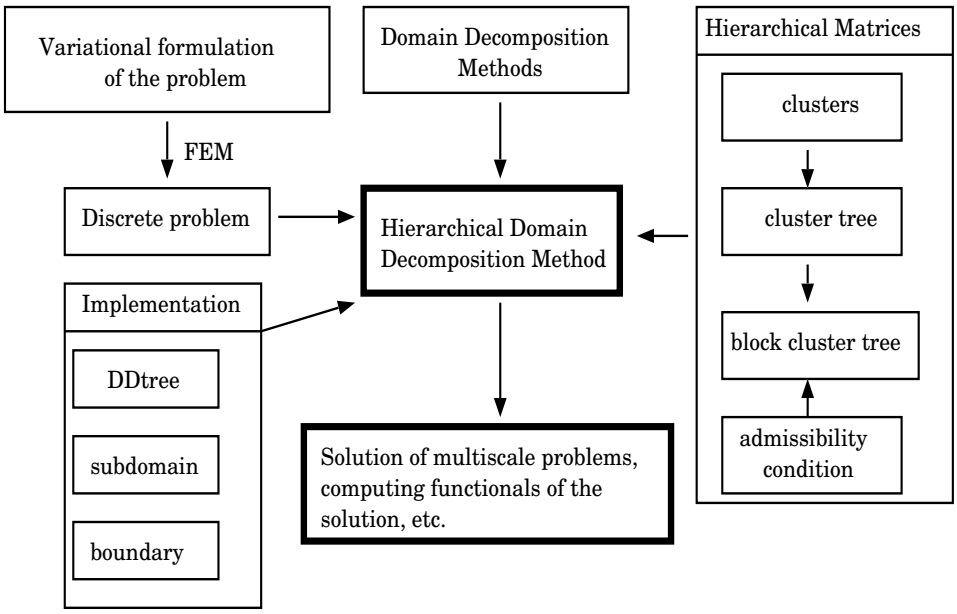


Figure 1.1: Diagram of the dissertation. $A \to B$ means that A is applied to B.

This dissertation is structured as follows:

Chapter 2. Multiscale Problems and Methods for their Solution

The L^∞ coefficients in (1.2) may exhibit multiscale (e.g., jumping as well as oscillatory) behaviour. We give a short introduction to multiscale problems and briefly describe well established methods for their solution. We consider the homogenization method and the multiscale finite element method (MsFEM). To explain the resonance effect which appears in MsFEM we provide the error estimates. At the end of the chapter we offer two examples of multiscale problems.

Chapter 3. Classical Theory of the FE Method

We describe important notation, methods and theorems from numerical analysis. We repeat the definitions of Sobolev spaces H^k, H_0^k and properties of these spaces. We give a brief introduction to the variational formulation of the initial problem. Then we explain how to apply the finite element method to get a system of linear equations. We also recall error estimates for the chosen triangulation and chosen basis functions.

Chapter 4. Hierarchical Domain Decomposition Method

The HDD method includes two recursive algorithms: "Leaves to Root" and "Root to Leaves". The first one creates a set of auxiliary boundary-to-boundary and domain-to-boundary mappings and then the set of main boundary-to-interface and

domain-to-interface mappings. The second algorithm applies the main mappings to compute the solution. There are three modifications of the HDD method: HDD with the right-hand side $f_h \in V_H \subset V_h$, HDD with truncation of the small scales and a combination of the first and the second modifications. One may see the comparison of HDD with truncation of the small scales with the known MsFEM method [40]. We show that HDD is appropriate for the problems with repeated patterns. In conclusion we show how to apply HDD to compute different functionals of the solution.

Chapter 5. Hierarchical Matrices

The hierarchical matrices ($\mathcal{H}$-matrices) have been used in a wide range of applications since their introduction in 1999 by Hackbusch [33]. They provide a format for the data-sparse representation of fully-populated matrices. The main idea in $\mathcal{H}$-matrices is the approximation of certain subblocks of a given matrix by low-rank matrices. At the beginning we present two examples of $\mathcal{H}$-matrices (see Fig. 5.1). Then we list the main steps which are necessary for building hierarchical matrices: construction of the cluster tree, choice of the admissibility condition and construction of the block cluster tree. We introduce the class of low-rank matrices $\mathcal{R}(k, n, m)$, where k, n, m are integers, $k \ll \min\{n, m\}$, and then the low-rank arithmetic. We briefly describe how to perform the hierarchical matrix operations efficiently (with almost linear complexity). It will be shown that the cost of the basic $\mathcal{H}$-matrix arithmetic (matrix-matrix addition, matrix-matrix multiplication, inversion of matrices) is not greater than $\mathcal{O}(n \log^{\alpha} n)$, $\alpha = 1, 2, 3$ (see Theorem 5.10.1). We then present two procedures for extracting a part of a hierarchical matrix (see Algorithm 5.9.3) and converting one $\mathcal{H}$-matrix format to another one. The last two procedures are needed for adding two hierarchical matrices with different block structures (see Lemma 5.10.8).

Chapter 6. Application of $\mathcal{H}$-Matrices to HDD

The exact matrix arithmetic in the HDD method can be replaced by the approximate $\mathcal{H}$-matrix arithmetic to improve efficiency. Here we will explain the construction of $\mathcal{H}$-matrix approximations for the domain-to-boundary (denoted by Ψ^f) and boundary-to-boundary (denoted by Ψ^g) mappings, which are essential for the definition of the HDD method. It will be shown that the boundary-to-interface mapping (denoted by Φ^g) can be approximated by a low-rank matrix and the domain-to-interface mapping (denoted by Φ^f) by an $\mathcal{H}$-matrix. Letting $\omega = \omega_1 \cup \omega_2$, where $\omega, \omega_1, \omega_2 \subset \Omega$, we also provide the algorithms for building Ψ^g_ω from $\Psi^g_{\omega_1}$ and $\Psi^g_{\omega_2}$ (see Algorithms 6.3.1 and 6.3.2) and the algorithms for building Ψ^f_ω from $\Psi^f_{\omega_1}$ and $\Psi^f_{\omega_2}$ (see Algorithms 6.3.3 and 6.3.4).

Chapter 7. Complexity and Storage Requirement of HDD

The HDD method consists of two algorithms "Leaves to Root" and "Root to Leaves". Using the costs of the standard $\mathcal{H}$-matrix operations, we estimate the

computational complexities of both algorithms. The first algorithm produces a set of domain-to-interface mappings and boundary-to-interface mappings. The second algorithm applies this set of mappings to compute the solution (only matrix-vector multiplications are required).

Let n_h, n_H be the respective numbers of degrees of freedom of the fine grid $\mathcal{T}_h$ and of the coarse grid $\mathcal{T}_H$. We prove that the complexities of the algorithms "Leaves to Root" and "Root to Leaves" are

$$\mathcal{O}(k^2 n_h \log^3 n_h) \quad \text{and} \quad \mathcal{O}(k n_h \log^2 n_h),$$

respectively (cf. Lemmas 7.2.3 and 7.3.3) and the storage requirement is $\mathcal{O}(k n_h \log^2 n_h)$ (see Lemma 7.3.4). As we show in Lemmas 7.4.4 and 7.4.3, the complexities of the same algorithms for the two-grid modification are

$$\mathcal{O}(k^2 \sqrt{n_h n_H} \log^3 \sqrt{n_h n_H}) \quad \text{and} \quad \mathcal{O}(k \sqrt{n_h n_H} \log \sqrt{n_h} \log \sqrt{n_H})$$

The storage requirement for the two-grid modification of HDD is (cf. Lemma 7.4.1)

$$\mathcal{O}(k \sqrt{n_h n_H} \log \sqrt{n_h} \log \sqrt{n_H}).$$

Chapter 8. Parallel Computing

We present the parallel HDD algorithm and estimate parallel complexities of the algorithms "Leaves to Root" and "Root to Leaves". We consider the parallel model, which consists of q processors and a global memory which can be accessed by all processors simultaneously. The communication time between processors is negligible in comparison with the computational time. For a machine with q processors the parallel complexity of the algorithm "Leaves to Root" is estimated (Lemma 8.3.3) by

$$\frac{C' k^2 \sqrt{n_h} \log^2 \sqrt{n_h} + \tilde{C} k^2 n_h}{q^{0.45}} + C''(1 - \frac{3^r}{4^r}) \sqrt{n_h} n_{min}^2 + C k^2 \frac{n_h}{q} \log^3 \frac{n_h}{q}$$

where $\tilde{C}, C', C'', C \in \mathbb{R}_+$.
The parallel complexity of the algorithm "Root to Leaves" on a machine with q processors is estimated (Lemma 8.3.6) by

$$C k^2 \frac{n_h}{q} \log^2 \frac{n_h}{q} + \frac{28 k \sqrt{n_h}}{q^{0.45}}, \quad C \in \mathbb{R}_+.$$

Chapter 9. Implementation of the HDD Package

The result of the implementation is a package of programs which uses the following libraries: HLIB, LAPACK, BLAS and external triangulators (for complex geometry). We present the data structures for the triangulation, the grid hierarchy and the HDD method. We describe the connection between the data structures "vertex", "triangle", "edge", "subdomain", "boundary" and "hierarchical decomposition tree". Then, we present the algorithms of the hierarchical decomposition and of the mesh refinement.

Four modifications of HDD (numbered by subindex) which require less computational resources than the original HDD were implemented. HDD_1 works with the right-hand side defined only on a coarse scale (see Section 4.3.6). HDD_2 computes the solution in all ω with $diam(\omega) \geq H$, and the mean value of the solution inside all ω with $diam(\omega) < H$. The mean value is a functional of the right-hand side and the Dirichlet data (see Section 4.4.5). HDD_3 is a combination of HDD_1 and HDD_2. HDD_4 is developed for problems with a homogeneous right-hand side (see Section 4.4.8).

Chapter 10. Numerical Results

We demonstrate numerical experiments to confirm the estimates of the $\mathcal{H}$-matrix approximation errors (Chapter 5), the computational times and the needed storage requirements (Chapter 7).

We demonstrate almost linear complexities of the algorithms "Leaves to Root" and "Root to Leaves". We also show an almost linear dependence of the memory requirement and the executing time on the number of degrees of freedoms. Next, we apply HDD to the problem with highly jumping coefficients (e.g., skin problem) and to problems with strong oscillatory coefficients, e.g.,

$$\alpha(x, y) = 2 + \sin(\nu \cdot x) \sin(\nu \cdot y),$$

where ν is the frequency (see Table 10.17).

The solution, obtained by the HDD method, is compared with the solutions obtained by the preconditioned conjugate gradient (PCG) method and the direct $\mathcal{H}$-Cholesky method. It is shown that the HDD method requires less memory than the direct $\mathcal{H}$-matrix inverse and a little bit more than the PCG method with $\mathcal{H}$-Cholesky preconditioner. But note that HDD computes the solution operators $\mathcal{F}_h$ and $\mathcal{G}_h$ in (1.4) whereas PCG only the solution. Other experiments demonstrate the possibility of obtaining a solution on a coarse scale and the possibility of truncation of the small scales. Finally, it will be shown that HDD is very efficient for problems with many right-hand sides.

2 Multiscale Problems and Methods for their Solution

2.1 Introduction

In the last years, we have seen large growth of activities in multiscale modeling and computating, with applications in material sciences, fluid mechanics, chemistry, biology, astronomy and other branches of science.

The basic set-up of a multiscale problem is as follows. We have a system whose microscopic behaviour, denoted by the state variable u, is described by a given microscopic model. This microscopic model is too expensive to be used in dense detail. Instead, we are interested in the macroscopic behaviour of the system. We want to use the microscopic model to extract all microscale details to build a good approximation for the macroscale behaviour of the system. Our purpose is not to solve dense microscale problems in detail, but to use a more efficient combined macro-micro modeling technique.

There has been a long history of studying multiscale problems in applied mathematics and computational science (see [7]). Multiscale problems are multiphysical in nature; namely, the processes at different scales are governed by physical laws of different characters: for example, quantum mechanics at one scale and classical mechanics at another. Well-known examples of problems with multiple length scales include turbulent flows, mass distribution in the universe, weather forecasting and ocean modeling. Another example is an elliptic equation with a highly oscillatory coefficient arising in material science or flow through porous media.

On the computational side, several numerical methods have been developed which address explicitly the multiscale nature of the solutions. These include the upscaling method ([21]), the averaging method, the homogenization method, the heterogeneous multiscale method [18], [4] the finite difference heterogeneous multiscale method [3] (see also [19], [20]).

Another interesting approach is offered in [39]. The authors consider an elliptic homogenization problems in a domain $\Omega \subset \mathbb{R}^d$ with $n+1$ separated scales and reduce it to elliptic one-scale problems in dimension $(n+1)d$.

Example 2.1.1 *An example in Fig. 2.1 shows the solution of a multiscale problem. On the fine scale we see a complex behaviour of the solution, but on the coarse scale the solution looks like the function $sin(x)$. For many practical applications, the fine properties of the solution are of no interest and it suffices to find the macro properties of the solution.*

Despite significant progress, purely analytical techniques are still very limited when it comes to problems of practical interest.

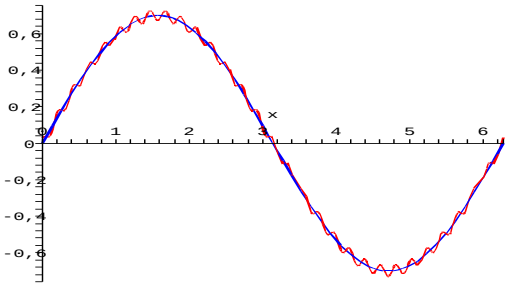


Figure 2.1: An example of a multiscale solution (wavy curve) and its macroscopic approximation.

2.2 Multiscale Methods

Homogenization

There exist a lot of composite materials with a large number of heterogeneties (inclusions or holes). One can try to characterise the properties of such a material locally, i.e., on the microscale. But in practice, it is much more important to know macroscale characteristics. In the frame of the heterogenization theory, the heterogeneous material is replaced by a fictitious one - the homogenized material. The behaviour of this homogenized material should be as close as possible to the behaviour of the composite itself. One tries to describe the global properties of the composite by taking into account the local properties of its constituents.
Homogenization is a way of extracting an equation for the coarse scale behaviour that takes into account the effect of small scales (see [12], [42]). The fine scales cannot be just ignored because the solution on the coarser scales depends on the fine scales. After homogenization the initial equation does not contain fine scales and is therefore much easier to solve.
For the periodic case, the homogenization process consists in replacing the initial partial differential equation with rapidly oscillating coefficients that describe the composite material by a partial differential equation with the fictitious homogenized coefficients. The homogenized coefficients are obtained by solving a non oscillating partial differential equation on a period of reference.
Let $\Omega = (0,1) \times (0,1)$ and $f \in L^2(\Omega)$. Let $\alpha \in L^\infty(\Omega)$ be a positive function such that

$$0 < \underline{\alpha} \leq \alpha(\frac{\mathbf{x}}{\varepsilon}) \leq \overline{\alpha} < +\infty,$$

where $\underline{\alpha}$ and $\overline{\alpha}$ are constants. We denote the nodal point in Ω by the bold shrift (e.g., $\mathbf{x}$, $\mathbf{y}$). Assume $\alpha = \alpha(\frac{\mathbf{x}}{\varepsilon})$ is periodic with period ε. ε characterizes the small scale of the problem. We assume $\alpha(\mathbf{y})$ is periodic in Y and smooth. The model problem is:

$$\begin{array}{cc} -\nabla\alpha(\mathbf{x})\nabla u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega. \end{array} \tag{2.1}$$

Definition 2.2.1 *We denote the volume average over Y as $\langle\cdot\rangle = \frac{1}{\|Y\|}\int_Y d\mathbf{y}$.*

For an analysis of this and many other equations see [12], [17].
With classical finite element methods, one can obtain a good approximation only if the mesh size h is smaller than the finest scale, i.e., $h \ll \varepsilon$. But the memory requirements and CPU time grow polynomially with h^{-1} and soon become too large. One of the homogenization methods is the so-called multiple-scale method. Recently there have been many contributions on multiscale numerical methods, including the papers [6] and [17]. It seeks an asymptotic expansion of u^ε of the form

$$u^\varepsilon(\mathbf{x}) = u_0(\mathbf{x}) + \varepsilon u_1(\mathbf{x}, \frac{\mathbf{x}}{\varepsilon}) - \varepsilon\theta_\varepsilon + \mathcal{O}(\varepsilon^2), \tag{2.2}$$

where $\frac{\mathbf{x}}{\varepsilon}$ is the fast variable. The value of u^ε at the point $\mathbf{x}$ depends on two scales. The first one corresponds to $\mathbf{x}$, which describe the position in Ω. The other scale corresponds to $\frac{\mathbf{x}}{\varepsilon}$, which describe the local position of the point. The first variable, $\mathbf{x}$, is called the macroscopic (or slow) variable. The second one, $\frac{\mathbf{x}}{\varepsilon}$, is called the microscopic (or rapidly oscillating) variable. u_0 is the solution of the homogenized equation

$$\nabla\alpha^*\nabla u_0 = f \text{ in } \Omega, \quad u_0 = 0 \text{ on } \partial\Omega, \tag{2.3}$$

α^* is the constant effective coefficient, given by (see Def. 2.2.1)

$$\alpha^*_{ij} = \langle\alpha_{ik}(\mathbf{y})(\delta_{kj} - \frac{\partial}{\partial y_k}\chi^j)\rangle,$$

and χ^j is the periodic solution of

$$\nabla_y\alpha(\mathbf{y})\nabla_\mathbf{y}\chi^j = \frac{\partial}{\partial y_i}\alpha_{ij}(\mathbf{y})$$

with zero mean, i.e., $\langle\chi^j\rangle = 0$. It is known that α^* is symmetric and positive definite. Moreover, we have

$$u_1(\mathbf{x}, \mathbf{y}) = -\chi^j\frac{\partial u_0}{\partial x_j}.$$

Since in general $u_1 \neq 0$ on $\partial\Omega$, the boundary condition $u|_{\partial\Omega} = 0$ is enforced through the first-order correction term θ_ε, which is given by

$$\nabla\alpha(\frac{\mathbf{x}}{\varepsilon})\nabla\theta_\varepsilon = 0 \text{ in } \Omega, \quad \theta_\varepsilon = u_1(\mathbf{x}, \frac{\mathbf{x}}{\varepsilon}) \text{ on } \partial\Omega.$$

Under certain smoothness conditions, one can also obtain point-wise convergence of u to u_0 as $\varepsilon \to 0$.The condition can be weakened if the convergence is considered in the $L^2(\Omega)$ space. In [41] the authors use the asymptotic structure (2.2) to reveal the subtle details of the multiscale method and obtain sharp error estimates.

Heterogeneous multiscale method and multiscale finite element method

The heterogeneous multiscale method (HMM) [18],[19], [4] and the multiscale finite element method (MsFEM) [15] have been developed during the last time for solving, in particular, elliptic problems with multiscale coefficients. Comparison of

these two methods is done in [50]. Three examples when HMM can fail, are illustrated in [22] pp.105-107.
The main idea behind the Multiscale Finite Element Method is to build the local behaviour of the differential operator into the basis functions in order to capture the small scale effect while having a relative coarse grid over the whole domain. This is done by solving the equation on each element to obtain the basis functions, rather than using the linear basis functions. In [55], the authors apply MsFEM to the singularly perturbed convection-diffusion equation with periodic as well as random coefficients. They also consider elliptic equations with discontinuous coefficients and non-smooth boundaries. Both methods (HMM and MsFEM) solve only a subclass of the common multiscale problem. For example, HMM works like a fine scale solver without scale separation or any other special assumptions of the problem. Both methods for problems without scale separation do not give an answer with reasonable accuracy. In [50] the authors show that MsFEM incurs an $\mathcal{O}(1)$ error if the specific details of the fine scale properties are not explicitly used. They show also that for problems without scale separation HMM and MsFEM may fail to converge. HMM offers substantially savings of cost (compared to solve the full fine scale problems) for problems with scale separation. The advantage of both methods is their parallelizability.

Resonance Effect in MsFEM

For more information see please the original [40]. The variational problem of (2.1) is to seek $u \in H_0^1(\Omega)$, such that

$$a(u,v) = f(v), \quad \forall v \in H_0^1(\Omega), \tag{2.4}$$

where

$$a(u,v) = \int_\Omega \alpha_{ij} \frac{\partial v}{\partial x_i} \frac{\partial u}{\partial x_j} dx \quad \textit{and } f(v) = \int_\Omega f v dx. \tag{2.5}$$

A finite element method is obtained by restricting the weak formulation (2.4) to a finite-dimensional subspace of $H_0^1(\Omega)$. Let an axi-parallel rectangular grid $\mathcal{T}_h$ be given (Fig. 3.1). In each element $K \in \mathcal{T}_h$, we define a set of nodal basis $\{\phi_K^i,\, i = 1, ..., d\}$ with d being the number of nodes of the element. Let $\mathbf{x}_i = (x_i, y_i)$ $(i = 1, ..., d)$ be the nodal points in K. We neglect the subscript K when bases in one element are considered. The function ϕ^i satisfies

$$\nabla \alpha(\mathbf{x}) \nabla \phi^i = 0 \text{ in } K \in \mathcal{T}_h. \tag{2.6}$$

Let $\mathbf{x}_j \in \overline{K}$ $(j = 1, ..., d)$ be the nodal points of K. As usual the author requires $\phi^i(\mathbf{x}_j) = \delta_{ij}$. One needs to specify the boundary conditions to make (2.6) a well-posed problem. The authors assume in [40] that the basis functions are continuous across the boundaries of the elements, so that $V_h = span\{\phi^i : i = 1, ..., N\} \subset H_0^1(\Omega)$. Then they rewrite the problem (2.4): find $u^h \in V_h$ such that

$$a(u^h, v) = f(v), \quad \forall v \in V_h. \tag{2.7}$$

In [40] the authors describe two methods how to set up the boundary conditions for (2.6). We do not describe these methods here because there are many other variants and this is technical. In [41] the authors proved the following result.

Theorem 2.2.1 *Let u be the solution of the model problem (2.1) and u^h the solution of the corresponding equation in weak form (2.7). Then, there exist positive constants C_1 and C_2 independent of ε and h, such that*

$$\|u - u^h\|_{1,\Omega} \leq C_1 h \|f\|_{0,\Omega} + C_2(\varepsilon/h)^{\frac{1}{2}}, \quad (\varepsilon < h). \tag{2.8}$$

Proof: see [15], [40], [55].
To prove (2.8) the authors use the fact that the base functions defined by (2.6) have the same asymptotic structure as that of u; i.e.,

$$\phi^i = \phi_0^i + \varepsilon\phi_1^i - \varepsilon\theta^i + ... \quad (i = 1, ..., d),$$

where ϕ_0^i, ϕ_1^i, and θ^i are defined similarly as u_0, u_1, and θ_ε (see (2.2)), respectively. Note that applying the standard finite element analysis to MsFEM gives an pessimistic estimate $\mathcal{O}(\frac{h}{\varepsilon})$ in the H^1 norm, which is only useful for $h \ll \varepsilon$. In [40] the authors prove that in the case of periodic structure the MsFEM method converges to the correct effective solution as $\varepsilon \to 0$ independent of ε.
The following L^2-norm error estimate

$$\|u - u^h\|_{0,\Omega} \leq C_1 h^2 \|f\|_{0,\Omega} + C_2\varepsilon + C_3\|u^h - u_0^h\|_{l^2(\Omega)}, \tag{2.9}$$

is obtained from (2.8) by using the standard finite element analysis (see [40]). Here u_0^h is the solution of (2.3), $C_i > 0$, $(i = 1, 2, 3)$ are constants and $\|u^h\|_{l^2(\Omega)} = (\sum_i u^h(\mathbf{x}_i)^2 h^2)^{1/2}$. It is also shown that $\|u^h - u_0^h\|_{l^2(\Omega)} = \mathcal{O}(\varepsilon/h)$. Thus, we have

$$\|u - u^h\|_{0,\Omega} = \mathcal{O}(h^2 + \varepsilon/h). \tag{2.10}$$

It is clear that when $h \sim \varepsilon$, the multiscale method attains large errors in both H^1 and L^2 norms. This fact is called the *resonance effect*, the effect between the grid scale h and the small scale ε. To learn more about the resonance effect see [40]. In the same work, the authors propose an over-sampling technique to remove the resonance effect. After application of this over-sampling technique, the convergence in L^2 norm is $\mathcal{O}(h^2 + \varepsilon|\log(h)|)$ for $\varepsilon < h$.

2.3 Applications

Below we briefly consider two examples of multiscale problems to show that usual numerical approaches are unsufficient and other efficient multiscale methods are required. We hope that the HDD method, offered in this work, after some modifications can be applied for solving such types of problems.

A multiple scale model for porous media

Very important in modeling porous media is the use of different length scales. Figure 2.2 shows an example of porous media consisting of different types of stones on two length scales. Figure (a) demonstrates macroscale (the order is 10 meters), figure (b) microscale (10^{-3} meters). On the large scale, we identify different types of sand (stones). On the microscale, grains and pore channels are visible. In the figure we see the transition zone from a fine sand to a coarse sand. The void space is supposed to be filled with water. The behaviour of the liquid flow is influenced by effects on these different length scales.
On each scale different physical processes are appearing and different mathematical equations, which describe this processes are being used. More about the solving of this problem see [8], [21].

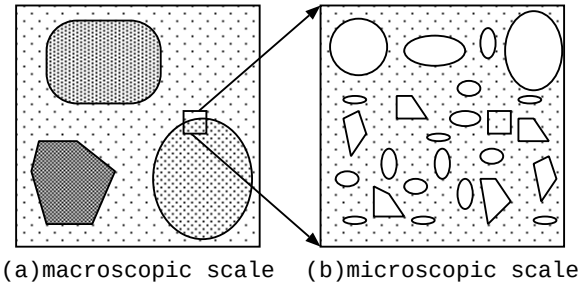


Figure 2.2: Two scales in a porous medium (see [8]).

A multiple scale model for tumor growth

In spite of the huge amount of resources that have been devoted to cancer research, many aspects remain obscure for experimentalists and clinicials, and many of the currently used therapeutic strategies are not entirely effective. One can divide the models for modeling cancer into two approaches: continuum models, mathematically formulated in terms of partial differential equations, and cellular automation (CA) models. Significant progress in developing mathematical models was done with the introduction of multiscale models. The tumor growth has an intrinsic multiple scale nature. It involves processes occurring over a variety of time and length scales: from the tissue scale to intracellular processes. The scheme of time and length scales is figured in Fig. 2.3. The multiscale tumor model include: blood flow, transport into the tissue of bloodborne oxygen, cell division, apoptosis etc. In the paper [5] the authors have proposed a multiple scale model for vascular tumor growth in which they have integrated phenomena at the tissue scale (vascular structural, adaptation,

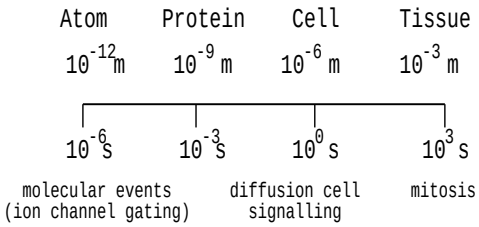


Figure 2.3: Example of time and length scales for modeling tumor growth.

and blood flow), cellular scale (cell-to-cell interaction), and the intracellular scale (cell-cycle, apoptosis). To get more details see [5].

3 Classical Theory of the FE Method

This Chapter contains classical results [31], [14]. We will hold on the original notation as in [31].

3.1 Sobolev Spaces

In this section we describe well known classical notation. Almost all material is taken from the book [31].
Let Ω be a open subset of $\mathbb{R}^n$. $L^2(\Omega)$ consists of all Lebesque-measurable functions whose squares on Ω are Lebesque-integrable.

3.1.1 Spaces $L^s(\Omega)$

$L^1(\Omega) = \{f : \Omega \to \mathbb{R} \text{ measurable } \mid \|f\|_{L^1(\Omega)} = \int_\Omega |f| dx < \infty\}$.
Let $1 < s < \infty$. Then

$$L^s(\Omega) = \{f : \Omega \to \mathbb{R} \mid |f|^s \in L^1, \|f\|_{L^s(\Omega)} = (\int_\Omega |f|^s dx)^{1/s} < \infty\}.$$

Let $s = \infty$ and $f : \Omega \to \mathbb{R}$ be measurable. Then define

$$\|f\|_{L^\infty(D)} := inf\{sup\{|f(x)| : x \in D \backslash A\} : \quad A \text{ is a set of measure zero }\}.$$

The definition of the space $L^\infty(\Omega)$ is:

$$L^\infty(\Omega) = \{f : \Omega \to \mathbb{R} \text{ measurable } \mid \|f\|_{L^\infty} < \infty\}$$

Theorem 3.1.1 *$L^2(\Omega)$ forms a Hilbert space with the scalar product*

$$(u, v)_0 := (u, v)_{L^2(\Omega)} := \int_\Omega u(x)\overline{v(x)}dx$$

and the norm

$$|u|_0 := \|u\|_{L^2(\Omega)} := \sqrt{\int_\Omega |u(x)|^2 dx}.$$

Definition 3.1.1 *$u \in L^2(\Omega)$ has a weak derivative $v := D^\alpha u \in L^2(\Omega)$ if for the latter $v \in L^2(\Omega)$ holds:*

$$(w, v)_0 = (-1)^{|\alpha|}(D^\alpha w, u)_0 \quad \textit{for all} \quad w \in C_0^\infty(\Omega).$$

3.1.2 Spaces $H^k(\Omega)$, $H_0^k(\Omega)$ and $H^{-1}(\Omega)$

Let $k \in \mathbb{N} \cup \{0\}$. Let $H^k(\Omega) \subset L^2(\Omega)$ be the set of all functions having weak derivatives $D^\alpha u \in L^2(\Omega)$ for $|\alpha| \leq k$:

$$H^k(\Omega) := \{u \in L^2(\Omega) : D^\alpha u \in L^2(\Omega) \quad \text{for} \quad |\alpha| \leq k\}.$$

Theorem 3.1.2 *$H^k(\Omega)$ forms a Hilbert space with the scalar product*

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

and the (Sobolev) norm

$$\|u\|_k := \|u\|_{H^k(\Omega)} := \sqrt{\sum_{|\alpha| \leq k} \|D^\alpha u\|^2_{L^2(\Omega)}}. \tag{3.1}$$

Definition 3.1.2 *The completion of $C_0^\infty(\Omega)$ in $L^2(\Omega)$ with respect to the norm (3.1) is denoted by $H_0^k(\Omega)$.*

Theorem 3.1.3 *$H_0^0(\Omega) = H^0(\Omega) = L^2(\Omega)$.*

Proof: see p.117 in [31].

The Sobolev space denoted here by $H^k(\Omega)$ is denoted by $W_2^k(\Omega)$ in other sources.

Definition 3.1.3 *$H^{-1}(\Omega)$ is the dual of $H_0^1(\Omega)$, i.e., $H^{-1}(\Omega) = \{f | f$ is a bounded linear functional on $H_0^1(\Omega)\}$.*

and the norm is

$$|u|_{-1} = \sup\{|(u, v)_{L^2(\Omega)}|/|v|_1 : 0 \neq v \in H_0^1(\Omega)\}.$$

3.2 Variational Formulation

Let us consider the following elliptic equation

$$Lu = f \quad \text{in} \quad \Omega, \tag{3.2}$$

$$L = \sum_{|\alpha| \leq m} \sum_{|\beta| \leq m} (-1)^{|\beta|} D^\beta a_{\alpha\beta}(\mathbf{x}) D^\alpha. \tag{3.3}$$

We assume the homogeneous Dirichlet boundary conditions

$$u = 0, \quad \frac{\partial u}{\partial n} = 0 \quad , ..., (\frac{\partial}{\partial n})^{m-1} u = 0 \quad \text{on} \quad \Gamma = \partial\Omega, \tag{3.4}$$

which are only meaningful if Γ is sufficiently smooth. Let $u \in C^{2m}(\Omega) \cap H_0^m(\Omega)$ be a classical solution of (3.2) and (3.4). To derive the variational formulation of (3.2)-(3.4) we multiply equation (3.2) by $v \in C_0^\infty(\Omega)$ and integrate the result over

the domain Ω.
Since $v \in C_0^\infty(\Omega)$, the integrand vanishes in the proximity of Γ. After integration by parts we obtain the variational formulation (the so-called 'weak' formulation):

$$\sum_{|\alpha|,|\beta|\leq m} \int_\Omega a_{\alpha\beta}(D^\alpha u(\mathbf{x}))(D^\beta v(\mathbf{x}))d\mathbf{x} = \int_\Omega f(\mathbf{x})v(\mathbf{x})d\mathbf{x} \tag{3.5}$$

$$\text{for all} \quad v \in C_0^\infty(\Omega).$$

Definition 3.2.1 *The bilinear form and the functional are*

$$a(u,v) := \sum_{|\alpha|,|\beta|\leq m} \int_\Omega a_{\alpha\beta}(D^\alpha u(\mathbf{x}))(D^\beta v(\mathbf{x}))d\mathbf{x}, \tag{3.6}$$

$$\varphi(v) := \int_\Omega f(\mathbf{x})v(\mathbf{x})d\mathbf{x}.$$

Theorem 3.2.1 *Let $a_{\alpha\beta} \in L^\infty(\Omega)$. The bilinear form defined by (3.6) is bounded on $H_0^m(\Omega) \times H_0^m(\Omega)$.*

Proof: see p.146 in [31].

Definition 3.2.2 *The variational formulation (or weak formulation) of the boundary value problem (3.2)-(3.4) is:*

$$\textit{find} \quad u \in H_0^m(\Omega) \quad \textit{with} \quad a(u,v) = \varphi(v) \quad \textit{for all} \quad v \in C_0^\infty(\Omega). \tag{3.7}$$

The existence and uniqueness of the solution in the weak form can be proved by the Lax-Milgram Lemma.

Theorem 3.2.2 *(Lax-Milgram lemma)*
Let V be a Hilbert space, let $a(\cdot,\cdot) : V \times V \to \mathbb{R}$ be a continuous V-elliptic bilinear form, and let $\varphi : V \to \mathbb{R}$ be a continuous linear form. Then the abstract variational problem: Find an element u such that

$$u \in V \quad \textit{and} \quad \forall v \in V, \quad a(u,v) = \varphi(v),$$

has one and only one solution.

Proof: see [16].

3.3 Inhomogeneous Dirichlet Boundary Conditions

Let us consider the boundary value problem

$$Lu = f \quad \textit{in} \quad \Omega, \quad u = g \quad \text{on} \quad \Gamma, \tag{3.8}$$

where L is a differential operator of second order. The variational formulation of the boundary value problem reads:

$$\text{find} \quad u \in H^1(\Omega) \quad \text{with} \quad u = g \quad \text{on} \quad \Gamma \quad \text{such that} \tag{3.9}$$

$$a(u,v) = \varphi(v) \quad \text{for all} \quad v \in H_0^1(\Omega). \tag{3.10}$$

Remark 3.3.1 *For the solvability of Problem (3.9),(3.10) it is necessary that:*

$$\text{there exists a function} \quad u_0 \in H^1(\Omega) \quad \text{with} \quad u_0|_\Gamma = g. \tag{3.11}$$

If such function u_0 is known, then we obtain the following weak formulation:

$$\text{Let} \quad u_0 \quad \text{satisfy (3.11)}; \quad \text{find} \quad w \in H_0^1(\Omega), \quad \text{such that} \tag{3.12}$$

$$a(w,v) = \tilde{\varphi}(v) := \varphi(v) - a(u_0,v) \quad \text{for all} \quad v \in H_0^1(\Omega). \tag{3.13}$$

The advantage of this formulation is that the functions w and v are from the same space $H_0^1(\Omega)$.

Remark 3.3.2 *In this work we assume that g and Ω are such that u_0 from the above remark exists.*

Theorem 3.3.1 *(Existence and uniqueness)*
Let the following problem

$$\textit{find} \quad u \in H_0^m(\Omega) \quad \textit{with} \quad a(u,v) = \varphi(v) \quad \textit{for all} \quad v \in H_0^m(\Omega) \tag{3.14}$$

(with homogeneous boundary values) be uniquely solvable for all $\varphi \in H^{-1}(\Omega)$. Then Condition (3.11) is sufficient, and necessary, for the unique solvability of Problem (3.9),(3.10).

Proof: see Theorem 7.3.5 in [31].

The variational formulation is the foundation of the Ritz-Galerkin discretisation method.

3.4 Ritz-Galerkin Discretisation Method

Suppose we have a boundary value problem in its variational formulation:

$$\text{Find} \quad u \in V, \quad \text{so that} \quad a(u,v) = \varphi(v) \quad \text{for all} \quad v \in V, \tag{3.15}$$

where we are thinking, in particular, of $V = H_0^m(\Omega)$ or $V = H^1(\Omega)$.
We assume that $a(\cdot,\cdot)$ is a bounded bilinear form defined on $V \times V$, and that φ is from the dual space V':

$$\begin{array}{ll} |a(u,v)| \le C_s \|u\|_V \|v\|_V & \text{for } u,v \in V, \quad C_s \in \mathbb{R}_+ \\ |\varphi(v)| \le C \|v\|_V & \text{for } v \in V, \quad C_s \in \mathbb{R}_+. \end{array} \tag{3.16}$$

The Ritz-Galerkin discretisation consists in replacing the infinite-dimensional space V with a finite-dimensional space V_N:

$$V_N \subset V, \quad dim V_N = N < \infty. \tag{3.17}$$

Since $V_N \subset V$, both $a(u,v)$ and $\varphi(v)$ are defined for $u,v \in V_N$. Thus, we pose the following problem:

$$\text{Find} \quad u^N \in V_N, \quad \text{so that} \quad a(u^N,v) = \varphi(v) \quad \text{for all} \quad v \in V_N. \tag{3.18}$$

Definition 3.4.1 *The solution of (3.18), if it exists, is called the Ritz-Galerkin solution (belonging to V_N) of the boundary value problem (3.15).*

To calculate a solution we need a basis of V_N. Let $\{b_1, b_2, ..., b_N\}$ be such a basis,

$$V_N = span\{b_1, ..., b_N\}. \tag{3.19}$$

For each coefficient vector $\mathbf{v} = \{v_1, ..., v_N\}^T$ we define

$$\mathbf{P} : \mathbb{R}^n \to V_N \subset V, \quad \mathbf{Pv} := \sum_{i=1}^{N} v_i b_i. \tag{3.20}$$

Thus, we can rewrite the problem (3.18):

$$\text{Find} \quad u^N \in V_N, \quad \text{so that} \quad a(u^N, b_i) = \varphi(b_i) \quad \text{for all} \quad i = 1, ..., N. \tag{3.21}$$

Proof: see [31], p.162.

We now seek $\mathbf{u} \in \mathbb{R}^N$ so that $u^N = \mathbf{Pu}$.

Theorem 3.4.1 *Assume (3.19). The $N \times N$-matrix $A = (A_{ij})$ and the N-vector $\mathbf{c} = (c_1, ..., c_N)^T$ are defined by*

$$A_{ij} := a(b_j, b_i) \qquad (i, j = 1, ..., N), \tag{3.22}$$

$$c_i := \varphi(b_i) \qquad (i = 1, ..., N), \tag{3.23}$$

Then the problem (3.18) and

$$A\mathbf{u} = \mathbf{c} \tag{3.24}$$

are equivalent.

If $\mathbf{u}$ is a solution of (3.24), then $u^N = \sum_{j=1}^{N} u_j b_j$ solves the problem (3.18). In the opposite direction, if u^N is a solution of (3.18), then $\mathbf{u} := \mathbf{P}^{-1} u^N$ is a solution of (3.24).
Proof: see [31], p.162.
The following theorem estimates the Ritz-Galerkin solution.

Theorem 3.4.2 *(Cea) Assume that (3.16),(3.17) and*

$$inf\{sup\{|a(u, v)| : v \in V_N, \|v\|_V = 1\} : u \in V_N, \|u\|_V = 1\} = \epsilon_N > 0 \tag{3.25}$$

hold. Let $u \in V$ be a solution of the problem (3.15), and let $u^N \in V_N$ be the Ritz-Galerkin solution of (3.18). Then the following estimate holds:

$$\|u - u^N\|_V \leq (1 + \frac{C_s}{\epsilon_N}) inf_{w \in V_N} \|u - w\|_V \tag{3.26}$$

with C_s from (3.16). Note that $inf_{w \in V_N} \|u - w\|_V$ is the distance from the function u to V_N.

Proof: see [31], p.168.

3.5 FE Method

3.5.1 Linear Finite Elements for $\Omega \subset \mathbb{R}^2$

The method of finite elements (FE) is very common for the numerical treatment of elliptic partial differential equations. This method is based on the variational formulation of the differential equation. Alternative methods to FE are finite difference methods and finite volume methods, but FE can be applied to the problems with more complicated geometry.

First, we partition the given domain Ω into (finitely many) subdomains (elements). In 2D problems we use triangles.

Definition 3.5.1 *A partition $\tau = \{T_1, T_2, ..., T_M\}$ of Ω into triangular elements is called admissible (see an example in Fig. 3.1) if the following conditions are fulfilled:*

1. *T_i $(1 \leq i \leq N)$ are open triangles.*
2. *$\overline{\Omega} = \cup_{i=1}^{M} \overline{T}_i$.*
3. *If $\overline{T}_i \cap \overline{T}_j$ consists of exactly one point, then it is a common vertex of T_i and T_j.*
4. *If for $i \neq j$, $\overline{T}_i \cap \overline{T}_j$ consists of more than one point, then $\overline{T}_i \cap \overline{T}_j$ is a common edge of T_i and T_j.*

Examples of inadmissible triangulations:

1. Two triangles have a common edge, but in one triangle this edge is smaller than in another.
2. The intersection of two triangles is not empty.

An inadmissible triangulation is not allowed, because it is not clear how to require continuity from one triangle to other.

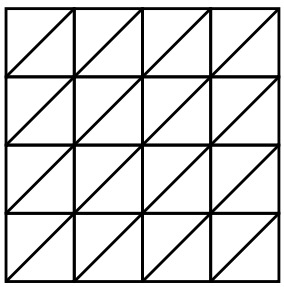


Figure 3.1: An example of an admissible triangulation.

Let τ be an admissible triangulation. The point $\mathbf{x}$ is called a **node** (of τ) if $\mathbf{x}$ is a vertex of one of the $T_i \in \tau$.
We define V_N as the subspace of the piecewise linear functions:

$$V_N := \{u \in C^0(\overline{\Omega}) : u = 0 \quad \text{on } \partial\Omega; \text{on each } T_i \in \tau \text{ the function } u \text{ agrees with a linear function, i.e., } u(x,y) = a_{i1} + a_{i2}x + a_{i3}y \text{ on } T_i\}. \tag{3.27}$$

Remark 3.5.1 *In the case of inhomogeneous boundary conditions, e.g.* $u = g$ *on* $\partial\Omega$ *we delete the requirement* $u = 0 \quad$ *on* $\partial\Omega$.

Let $\mathbf{x}^i$ $(1 \le i \le N)$ be the nodes of τ. For an arbitrary set of numbers $\{u_i\}_{i=1..N}$ there exists exactly one $u \in V_N$ with $u(\mathbf{x}^i) = u_i$. It may be written as $u = \sum_{i=1}^{N} u_i b_i$, where the basis functions b_i are characterised by

$$b_i(\mathbf{x}^i) = 1, \quad b_i(\mathbf{x}^j) = 0 \quad j \neq i. \tag{3.28}$$

If $T \in \tau$ is a triangle with the vertices $\mathbf{x}_i = (x_i, y_i)$ and $\mathbf{x}' = (x', y')$, $\mathbf{x}'' = (x'', y'')$, then

$$b_i(x, y) = \frac{(x - x')(y'' - y') - (y - y')(x'' - x')}{(x_i - x')(y'' - y') - (y_i - y')(x'' - x')} \quad \text{on } T. \tag{3.29}$$

We are interested in the bilinear form, associated to the initial equation:

$$a(u, v) = \int_\Omega \alpha(\mathbf{x}) \langle \nabla u, \nabla v \rangle d\mathbf{x}. \tag{3.30}$$

The coefficients of the stiffness matrix A are computed by the following formula:

$$A_{ij} = a(b_j, b_i) = \sum_k \int_{T_k} \alpha(\mathbf{x}) \langle \nabla b_j, \nabla b_i \rangle d\mathbf{x}. \tag{3.31}$$

1. If $i = j$, we have to integrate by all triangles which meet the vertex $\mathbf{x}^i$.
2. If $i \neq j$, we have to integrate by all triangles which contain both vertices $\mathbf{x}^i$ and $\mathbf{x}^j$.
3. $A_{ij} = 0$ if $\mathbf{x}^i$ and $\mathbf{x}^j$ are not directly connected by the side of a triangle.

Thus, we obtain a data-sparse matrix.

Example 3.5.1 *If we choose the basis functions as in (3.29) and the bilinear form by* $a(u, v) = \int_\Omega \langle \nabla u, \nabla v \rangle d\mathbf{x}$, *then for inner nodes*

$$L_{ii} = 4, \quad A_{ij} = -1 \quad \mathbf{x}_i - \mathbf{x}_j = (0, \pm h) \quad or \quad (\pm h, 0), \quad A_{ij} = 0 \quad otherwise; \tag{3.32}$$

Remark 3.5.2 *To calculate* $A_{ij} = \int_{T_k} \alpha(\mathbf{x}) \langle \nabla b_j, \nabla b_i \rangle d\mathbf{x}$ *numerically we use the basic three points quadrature formula on a triangle (see Table 3.1).*
If $b_i \in P^1$, *then* $\nabla b_i = const, \nabla b_j = const$ *and the coefficients* A_{ij} *are*

$$\begin{gathered} A_{ij} = \int_{T_k} \alpha(\mathbf{x}) \langle \nabla b_j, \nabla b_i \rangle d\mathbf{x} = \langle \nabla b_j, \nabla b_i \rangle \cdot \sum_{k=1}^{3} \alpha(v_k) w_k, \quad where \\ v_k = v_k(x, y) \quad from\ (3.35), \quad w_k \quad from\ Tables\ (3.1), (3.2) \end{gathered} \tag{3.33}$$

i	weights w_i	d_{i1}	d_{i2}	d_{i3}
1	0.33(3)	0.5	0.5	0.0
2	0.33(3)	0.0	0.5	0.5
3	0.33(3)	0.5	0.0	0.5

Table 3.1: The coefficients of the basic 3-point quadrature rule for a triangle (used in (3.35) and (3.33)). This rule calculates exactly the value of integrals for polynomial degree 2 (see [16], [54]).

Remark 3.5.3 *We compute the FE right-hand side* $\mathbf{c}$ *by the following formula:*

$$c_j := \int_{\operatorname{supp} b_j} f b_j d\mathbf{x}, \tag{3.34}$$

where $j = 1, ..., N$ *and* f *is the right-hand side in (3.8). For non-zero Dirichlet boundary data see (3.13).*

Remark 3.5.4 *It makes sense to apply 12-point quadrature rule if the discretisation error is smaller than the quadrature error. If the discretisation error is larger than the quadrature error, it is reasonable to apply the 3-point quadrature rule.*

i	weights w_i	d_{i1}	d_{i2}	d_{i3}
1	0.050844906370207	0.873821971016996	0.063089014491502	0.063089014491502
2	0.050844906370207	0.063089014491502	0.873821971016996	0.063089014491502
3	0.050844906370207	0.063089014491502	0.063089014491502	0.873821971016996
4	0.116786275726379	0.501426509658179	0.249826745170910	0.249826745170910
5	0.116786275726379	0.249826745170910	0.501426509658179	0.249826745170910
6	0.116786275726379	0.249826745170910	0.249826745170910	0.501426509658179
7	0.082851075618374	0.636502499121399	0.310352451033785	0.053145049844816
8	0.082851075618374	0.636502499121399	0.053145049844816	0.310352451033785
9	0.082851075618374	0.310352451033785	0.636502499121399	0.053145049844816
10	0.082851075618374	0.310352451033785	0.053145049844816	0.636502499121399
11	0.082851075618374	0.053145049844816	0.310352451033785	0.636502499121399
12	0.082851075618374	0.053145049844816	0.636502499121399	0.310352451033785

Table 3.2: The coefficients of the basic 12-point quadrature rule for a triangle (used in (3.35) and (3.33)). This rule calculates exactly the value of integrals for polynomial degree 6 (see [16], [54]).

If $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ are coordinates of the vertices of triangle, then we define the new quadrature points:

$$v_i(x, y) := (d_{i1}x_1 + d_{i2}x_2 + d_{i3}x_3, d_{i1}y_1 + d_{i2}y_2 + d_{i3}y_3), \quad i = 1, 2, 3, \tag{3.35}$$

where the coefficients d_{ij} are defined in Table 3.2.

3.5.2 Error Estimates for Finite Element Methods

In this subsection we use the notation u_h besides u^N in (3.18).
We suppose that

$$\begin{array}{ll} \tau & \text{is an admissible triangulation of} \quad \Omega \subset \mathbb{R}^2, \\ V_N & \text{is defined by (3.27)}, \quad \text{if } V = H_0^1(\Omega), \\ V_N & \text{is as in Remark (3.5.1)}, \quad \text{if } V = H^1(\Omega), \end{array} \tag{3.36}$$

There are two important theorems which allow to estimate $|u - u_h|$.

Theorem 3.5.1 *Assume that conditions (3.36) hold for τ, V_N, and V. Let α_0 be the smallest interior angle of all $T_i \in \tau$, while h is the maximum length of the sides of all $T_i \in \tau$. Then*

$$inf_{v \in V_N} \|u - v\|_{H^k(\Omega)} \leq C'(\alpha_0) h^{2-k} \|u\|_{H^2(\Omega)} \quad \text{for } k = 0, 1 \quad \text{and all } u \in H^2(\Omega) \cap V. \tag{3.37}$$

Proof: see [31], pp.187-188.
For the next theorem we need a new regularity condition on the adjoint problem to (3.15).

Definition 3.5.2 *The following problem is called the **adjoint problem** to (3.15).*

$$\text{Find} \quad u \in V, \quad \text{so that} \quad a^*(u, v) = \varphi(v) \quad \text{for all} \quad v \in V, \tag{3.38}$$

where $a^*(u, v) := a(v, u)$.

The new regularity condition is:

$$\begin{array}{c} \text{For each } \varphi \in L^2(\Omega) \quad \text{the problem (3.38)} \\ \text{has a solution } u \in H^2(\Omega) \cap V \quad \text{with } |u|_2 \leq C_R |\varphi|_0. \end{array} \tag{3.39}$$

Theorem 3.5.2 *(Aubin-Nitsche)*
Assume (3.39), (3.16),

$$inf\{sup\{|a(u, v)| : v \in V_N, \|v\|_V = 1\} : u \in V_N, \|u\|_V = 1\} = \epsilon_N \geq \tilde{\epsilon} > 0, \tag{3.40}$$

and

$$inf_{v \in V_N} |u - v|_1 \leq C_0 h |u|_2 \text{ for all } u \in H^2(\Omega) \cap V. \tag{3.41}$$

Let the problem (3.15) have the solution $u \in V$. Let $u_h \in V_N \subset V$ be the finite-element solution and V_N is the space of finite elements of an admissible triangulation. With a constant C_1 independent of u and h, we get:

$$|u - u_h|_0 \leq C_1 h |u|_1. \tag{3.42}$$

If the solution u also belongs to $H^2(\Omega) \cap V$, then there is a constant C_2, independent of u and h, such that

$$|u - u_h|_0 \leq C_2 h^2 |u|_2. \tag{3.43}$$

Proof: see [31], pp.190-191.

4 Hierarchical Domain Decomposition Method

The idea of the HDD method belongs to Hackbusch ([34]). This Chapter contains the main results of this work: the HDD method and its modifications.

4.1 Introduction

We repeat the initial boundary value problem to be solved:

$$\begin{cases} -\sum\limits_{i,j=1}^{2} \frac{\partial}{\partial_j}\alpha_{ij}\frac{\partial}{\partial_i}u = f & \text{in } \Omega \subset \mathbb{R}^2, \\ u = g & \text{on } \partial\Omega, \end{cases} \tag{4.1}$$

with $\alpha_{ij} = \alpha_{ji}(\mathbf{x}) \in L^\infty(\Omega)$ such that the matrix function $\mathcal{A}(\mathbf{x}) = (\alpha_{ij})_{i,j=1,2}$ satisfies $0 < \underline{\lambda} \le \lambda_{\min}(\mathcal{A}(\mathbf{x})) \le \lambda_{\max}(\mathcal{A}(\mathbf{x})) \le \overline{\lambda}$ for all $\mathbf{x} \in \Omega \subset \mathbb{R}^2$.
After a FE discretisation we obtain a system of linear equations $A\mathbf{u} = \mathbf{c}$.
In the past, several methods have been developed to combine the $\mathcal{H}$-matrix technique with the domain decomposition (DD) method (see [35], [38], [47]).

In [35] Hackbusch applies $\mathcal{H}$-matrices to the direct domain decomposition (DDD) method to compute A^{-1}. In this method he decomposes the initial domain Ω into p subdomains (proportional to the number of parallel processors). The respective stiffness matrix $A \in \mathbb{R}^{I\times I}$, $I := I(\overline{\Omega})$, is represented in the block-matrix form:

$$A = \begin{pmatrix} A_{11} & 0 & \dots & 0 & A_{1\Sigma} \\ 0 & A_{22} & \dots & 0 & A_{2\Sigma} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & A_{pp} & A_{p\Sigma} \\ A_{\Sigma 1} & A_{\Sigma 2} & \dots & A_{\Sigma p} & A_{\Sigma\Sigma} \end{pmatrix}. \tag{4.2}$$

Here $A_{ii} \in \mathbb{R}^{I_i\times I_i}$, where I_i is the index set of interior degrees of freedom in Ω_i. $A_{i\Sigma} \in \mathbb{R}^{I_i\times I_\Sigma}$, where $I_\Sigma := I \setminus \cup_{i=1}^{p} I_i$ is the index set of the degrees of freedom on the interface. E.g., in the case of finite elements. Assume that A and all submatrices A_{ii} are invertible, i.e., the subdomain problems are solvable. Let $S := A_{\Sigma\Sigma} - \sum_{i=1}^{p} A_{\Sigma i}A_{ii}^{-1}A_{i\Sigma}$. Then the inverse of A^{-1} can be defined by the following formula:

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & A_{pp}^{-1} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} A_{11}^{-1}A_{1\Sigma} \\ \vdots \\ A_{pp}^{-1}A_{p\Sigma} \\ -I \end{bmatrix} [S^{-1}A_{\Sigma 1}A_{11}^{-1}, \cdots, S^{-1}A_{\Sigma p}A_{pp}^{-1}, -S^{-1}]. \tag{4.3}$$

This DDD method is easily parallelizable. Let d be the spatial dimension. Then the complexity will be

$$\mathcal{O}(\frac{n}{p}\log np) + \mathcal{O}(p^{1/d}n^{d-1/d}), \quad \text{or}$$

$$\mathcal{O}(n^{d/d+1}\log n^{d/d+1}), \quad \text{for} \quad p = \mathcal{O}(1/d+1).$$

In [38], the authors start with the representation (4.2), apply the so-called direct Schur complement method, which is based on the $\mathcal{H}$-matrix technique and the domain decomposition method to compute the inverse of the Schur complement matrix associated with the interface. Building the approximate Schur complements corresponding to the subdomains Ω_i costs $\mathcal{O}(N_\Omega \log^q N_\Omega)$ for an FEM discretisation, where N_Ω is the number of degrees of freedom in the domain Ω.

In [47] the authors introduce the so-called $\mathcal{H}$-LU factorization which is based on the nested dissection method [24]. The initial domain Ω is decomposed hierarchically into three parts: Ω_{left}, γ and Ω_{right}, such that

$$\Omega_{left} \cap \Omega_{right} = \varnothing \text{ and } \Omega_{left} \cup \Omega_{right} \cup \gamma = \Omega.$$

Such a decomposition yields a block structure in which large off-diagonal subblocks of the finite element stiffness matrix are zero and remain zero during the computation of the $\mathcal{H}$-LU factorization. In this approach the authors compute the solution $\mathbf{u}$ as follows $\mathbf{u} = U^{-1}L^{-1}\mathbf{c}$, where the factors L and U are approximated in the $\mathcal{H}$-matrix format.

The HDD method, unlike the methods described above, has the capability to compute the solution on different scales retaining the information from the finest scales. HDD in the one-scale settings performs comparable to the methods from [35], [38], [47] with regards to the computational complexity.

4.2 Idea of the HDD Method

After Galerkin FE discretisation of (4.1) we construct the solution in the following form

$$u_h(f_h, g_h) = \mathcal{F}_h f_h + \mathcal{G}_h g_h, \tag{4.4}$$

where $u_h(f_h, g_h)$ is the FE solution of the initial problem (1.1), f_h the FE right-hand side, and g_h the FE Dirichlet boundary data. The hierarchical domain decomposition (HDD) method computes both operators $\mathcal{F}_h$ and $\mathcal{G}_h$. .

Domain decomposition tree ($T_{\mathcal{T}_h}$)

Let $\mathcal{T}_h$ be a triangulation of Ω. First, we decompose hierarchically the given domain Ω (cf. [24]). The result of the decomposition is the hierarchical tree $T_{\mathcal{T}_h}$ (see Fig. 4.1). The properties of the $T_{\mathcal{T}_h}$ are:

- Ω is the root of the tree,
- $T_{\mathcal{T}_h}$ is a binary tree,

- If $\omega \in T_{\mathcal{T}_h}$ has two sons $\omega_1, \omega_2 \in T_{\mathcal{T}_h}$, then
  $\omega = \omega_1 \cup \omega_2$ and ω_1, ω_2 have no interior point in common,
- $\omega \in T_{\mathcal{T}_h}$ is a leaf, if and only if $\omega \in \mathcal{T}_h$.

The construction of $T_{\mathcal{T}_h}$ is straight-forward by dividing Ω recursively in pieces. For practical purposes, the subdomains ω_1, ω_2 must both be of size $\approx |\omega|/2$ and the internal boundary

$$\gamma_\omega := \partial\omega_1 \backslash \partial\omega = \partial\omega_2 \backslash \partial\omega \tag{4.5}$$

must not be too large (see Fig. 4.2).

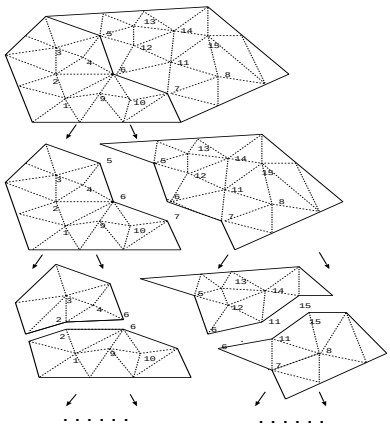


Figure 4.1: An example of the hierarchical domain decomposition tree $T_{\mathcal{T}_h}$.

Set of nodal points

Let $I := I(\overline{\Omega})$ and x_i, $i \in I$, be the set of all nodal points in $\overline{\Omega}$ (including nodal points on the boundary). We define $I(\omega)$ as a subset of I with $x_i \in \omega = \overline{\omega}$. Similarly, we define $I(\overset{\circ}{\omega})$, $I(\Gamma_\omega)$, $I(\gamma_\omega)$, where $\Gamma_\omega := \partial\omega$, $\overset{\circ}{\omega} = \omega \backslash \partial\omega$, for the interior, for the external boundary and for the internal boundary.

Idea

We are interested in computing the discrete solution u_h of (4.1) in Ω. This is equivalent to the computation of u_h on all γ_ω, $\omega \in T_{\mathcal{T}_h}$, since $I(\Omega) = \cup_{\omega \in T_{\mathcal{T}_h}} I(\gamma_\omega)$. These computations are performed by using the linear mappings Φ^f_ω, Φ^g_ω defined for all $\omega \in T_{\mathcal{T}_h}$. The mapping $\Phi^g_\omega : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\gamma_\omega)}$ maps the boundary data defined on $\partial\omega$ to the data defined on the interface γ_ω. $\Phi^f_\omega : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\gamma_\omega)}$ maps the right-hand side data defined on ω to the data defined on γ_ω.

Notation 4.2.1 *Let $g_\omega := u|_{I(\partial\omega)}$ be the local Dirichlet data and $f_\omega := f|_{I(\omega)}$ be the local right-hand side.*

The final aim is to compute the solution u_h along γ_ω in the form $u_h|_{\gamma_\omega} = \Phi^f_\omega f_\omega + \Phi^g_\omega g_\omega$, $\omega \in T_{\mathcal{T}_h}$. For this purpose HDD builds the mappings $\Phi_\omega := (\Phi^g_\omega, \Phi^f_\omega)$, for all

$\omega \in T_{\mathcal{T}_h}$. For computing the mapping Φ_ω, $\omega \in T_{\mathcal{T}_h}$, we, first, need to compute the auxiliary mapping $\Psi_\omega := (\Psi^g_\omega, \Psi^f_\omega)$ which will be defined later.

Thus, the HDD method consists of two steps: the first step is the construction of the mappings Φ^g_ω and Φ^f_ω for all $\omega \in T_{\mathcal{T}_h}$. The second step is the recursive computation of the solution u_h. In the second step HDD applies the mappings Φ^g_ω and Φ^f_ω to the local Dirichlet data g_ω and to the local right-hand side f_ω.

Notation 4.2.2 *Let $\omega \in T_{\mathcal{T}_h}$ and*

$$d_\omega := \Big((f_i)_{i\in I(\omega)}, (g_i)_{i\in I(\partial_\omega)} \Big) = (f_\omega, g_\omega) \tag{4.6}$$

be a composed vector consisting of the right-hand side from (4.1) restricted to ω and the Dirichlet boundary values $g_\omega = u_h|_{\partial\omega}$ (see also Notation 4.2.1).

Note that g_ω coincides with the global Dirichlet data in (4.1) only when $\omega = \Omega$. For all other $\omega \in T_{\mathcal{T}_h}$ we compute g_ω in (4.6) by the algorithm "Root to Leaves" (see Section 4.3.5).

Assuming that the boundary value problem (4.1) restricted to ω is solvable, we can define the local FE solution by solving the following discrete problem in the variational form (see (3.7)):

$$\begin{cases} a_\omega(U_\omega, b_j) = (f_\omega, b_j)_{L^2(\omega)}, & \forall\, j \in I(\mathring{\omega}), \\ U_\omega(\mathbf{x}_j) = g_j, & \forall\, j \in I(\partial\omega). \end{cases} \tag{4.7}$$

Here, b_j is the P^1-Lagrange basis function (see (3.29)) at $\mathbf{x}_j$ and $a_\omega(\cdot,\cdot)$ is the bilinear form (see (3.30)) with integration restricted to ω and $(f_\omega, b_j) = \int\limits_\omega f_\omega\, b_j\, d\mathbf{x}$.

Let $U_\omega \in V_h$ be the solution of (4.7) in ω. The solution U_ω depends on the Dirichlet data on $\partial\omega$ and the right-hand side in ω. Dividing problem (4.7) into two subproblems (4.8) and (4.9), we obtain $U_\omega = U^f_\omega + U^g_\omega$, where U^f_ω is the solution of

$$\begin{cases} a_\omega(U^f_\omega, b_j) = (f_\omega, b_j)_{L^2(\omega)}, & \forall\, j \in I(\mathring{\omega}), \\ U^f_\omega(\mathbf{x}_j) = 0, & \forall\, j \in I(\partial\omega) \end{cases} \tag{4.8}$$

and U^g_ω is the solution of

$$\begin{cases} a_\omega(U^g_\omega, b_j) = 0, & \forall\, j \in I(\mathring{\omega}), \\ U^g_\omega(\mathbf{x}_j) = g_j, & \forall\, j \in I(\partial\omega). \end{cases} \tag{4.9}$$

If $\omega = \Omega$ then (4.7) is equivalent to the initial problem (4.1) in the weak formulation.

4.2.1 Mapping $\Phi_\omega = (\Phi^g_\omega, \Phi^f_\omega)$

We consider $\omega \in T_{\mathcal{T}_h}$ with two sons ω_1, ω_2. Recall that we used γ_ω to denote the interface in ω (see (4.5)). Considering once more the data d_ω from (4.6), U^f_ω from (4.8) and U^g_ω from (4.9), we define $\Phi^f_\omega(f_\omega)$ and $\Phi^g_\omega(g_\omega)$ by

$$\big(\Phi^f_\omega(f_\omega)\big)_i := U^f_\omega(\mathbf{x}_i) \quad \forall i \in I(\gamma_\omega) \tag{4.10}$$

and

$$\left(\Phi^g_\omega(g_\omega)\right)_i := U^g_\omega(\mathbf{x}_i) \quad \forall i \in I(\gamma_\omega). \tag{4.11}$$

Since $U_\omega = U^f_\omega + U^g_\omega$, we obtain

$$\left(\Phi_\omega(d_\omega)\right)_i := \Phi^g_\omega(g_\omega) + \Phi^f_\omega(f_\omega) = U^f_\omega(\mathbf{x}_i) + U^g_\omega(\mathbf{x}_i) = U_\omega(\mathbf{x}_i) \tag{4.12}$$

for all $i \in I(\gamma_\omega)$.
Hence, $\Phi_\omega(d_\omega)$ is the trace of U_ω on γ_ω. Definition in (4.12) says that if the data d_ω are given then Φ_ω computes the solution of (4.7). Indeed, $\Phi_\omega d_\omega = \Phi^g g_\omega + \Phi^f f_\omega$.
Note that the solution u_h of the initial global problem coincide with U_ω in ω, i.e., $u_h|_\omega = U_\omega$.

4.2.2 Mapping $\Psi_\omega = (\Psi^g_\omega, \Psi^f_\omega)$

Let us, first, define the mappings Ψ^f_ω from (4.8) as

$$\left(\Psi^f_\omega(d_\omega)\right)_{i\in I(\partial\omega)} := a_\omega(U^f_\omega, b_i) - (f_\omega, b_i)_{L^2(\omega)}\,, \tag{4.13}$$

where $U^f_\omega \in V_h$, $U^f_\omega|_{\partial\omega} = 0$ and

$$a(U^f_\omega, b_i) - (f, b_i) = 0, \quad \text{for } \forall i \in I(\overset{\circ}{\omega}).$$

Second, we define the mapping Ψ^g_ω from (4.9) by setting

$$\left(\Psi^g_\omega(d_\omega)\right)_{i\in I(\partial\omega)} := a_\omega(U^g_\omega, b_i) - (f_\omega, b_i)_{L^2(\omega)} = a_\omega(U^g_\omega, b_i) - 0 = a_\omega(U^g_\omega, b_i), \tag{4.14}$$

where $U^g_\omega \in V_h$ and $\left(\Psi^g_\omega(d_\omega)\right)_i = 0$ for $\forall i \in I(\overset{\circ}{\omega})$.

The linear mapping Ψ_ω, which maps the data d_ω given by (4.6) to the boundary data on $\partial\omega$, is given in the component form as

$$\Psi_\omega(d_\omega) = \left(\Psi_\omega(d_\omega)\right)_{i\in I(\partial\omega)} := a_\omega(U_\omega, b_i) - (f_\omega, b_i)_{L^2(\omega)}\,. \tag{4.15}$$

By definition and (3.11)-(3.13), Ψ_ω is linear in (f_ω, g_ω) and can be written as $\Psi_\omega(d_\omega) = \Psi^f_\omega f_\omega + \Psi^g_\omega g_\omega$. Here U_ω is the solution of the local problem (4.7) and it coincides with the global solution on $I(\omega)$.

4.2.3 Φ_ω and Ψ_ω in terms of the Schur Complement Matrix

Let the linear system $A\mathbf{u} = F\mathbf{c}$ for $\omega \in T_{\mathcal{T}_h}$ be given. In Sections 4.3.1 and 4.3.3 we explain how to obtain the matrices A and F. A is the stiffness matrix for the domain $\overline{\omega}$ after elimination of the unknowns corresponding to $I(\overset{\circ}{\omega}\setminus\gamma_\omega)$. The matrix F comes from the numerical integration.
We will write for simplicity γ instead of γ_ω. Thus, $A : \mathbb{R}^{I(\partial\omega\cup\gamma)} \to \mathbb{R}^{I(\partial\omega\cup\gamma)}$, $\mathbf{u} \in \mathbb{R}^{I(\partial\omega\cup\gamma)}$, $F : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\partial\omega\cup\gamma)}$ and $\mathbf{c} \in \mathbb{R}^{I(\omega)}$. Decomposing the unknown vector $\mathbf{u}$ into two components $\mathbf{u}_1 \in \mathbb{R}^{I(\partial\omega)}$ and $\mathbf{u}_2 \in \mathbb{R}^{I(\gamma)}$, obtain

$$\mathbf{u} = \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix}.$$

The component $\mathbf{u}_1$ corresponds to the boundary $\partial\omega$ and the component $\mathbf{u}_2$ to the interface γ. Then the equation $A\mathbf{u} = F\mathbf{c}$ becomes

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} \mathbf{c}, \tag{4.16}$$

where

$$\begin{aligned} A_{11} : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\partial\omega)}, \quad A_{12} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\partial\omega)}, \\ A_{21} : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\gamma)}, \quad A_{22} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\gamma)}, \\ F_1 : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\partial\omega)}, \quad F_2 : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\gamma)}. \end{aligned}$$

The elimination of the internal points is done as it is shown in (4.17). To eliminate the variables $\mathbf{u}_2$, we multiply both sides of (4.16) by $A_{12}A_{22}^{-1}$, subtract the second row from the first, and obtain

$$\begin{pmatrix} A_{11} - A_{12}A_{22}^{-1}A_{21} & 0 \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \begin{pmatrix} F_1 - A_{12}A_{22}^{-1}F_2 \\ F_2 \end{pmatrix} \mathbf{c}. \tag{4.17}$$

We rewrite the last system as two equations

$$\begin{aligned} \tilde{A}\mathbf{u}_1 := (A_{11} - A_{12}A_{22}^{-1}A_{21})\mathbf{u}_1 = (F_1 - A_{12}A_{22}^{-1}F_2)\mathbf{c}, \\ A_{22}\mathbf{u}_2 = F_2\mathbf{c} - A_{21}\mathbf{u}_1. \end{aligned} \tag{4.18}$$

The unknown vector $\mathbf{u}_2$ is computed as follows

$$\mathbf{u}_2 = A_{22}^{-1}F_2\mathbf{c} - A_{22}^{-1}A_{21}\mathbf{u}_1. \tag{4.19}$$

Equation (4.19) is analogous to (4.43). The explicit expressions for the mappings Ψ_ω and Φ_ω follow from (4.18):

$$\Psi_\omega^g := A_{11} - A_{12}A_{22}^{-1}A_{21}, \tag{4.20}$$

$$\Psi_\omega^f := F_1 - A_{12}A_{22}^{-1}F_2, \tag{4.21}$$

$$\Phi_\omega^g := -A_{22}^{-1}A_{21}, \tag{4.22}$$

$$\Phi_\omega^f := A_{22}^{-1}F_2. \tag{4.23}$$

4.3 Construction Process

4.3.1 Initialisation of the Recursion

Our purpose is to get, for each triangle $\omega \in \mathcal{T}_h$, the system of linear equations

$$A \cdot \mathbf{u} = \tilde{\mathbf{c}} := F \cdot \mathbf{c},$$

where A is the stiffness matrix, $\mathbf{c}$ the discrete values of the right-hand side in the nodes of ω and F will be defined later. The matrix coefficients A_{ij} are computed by the formula

$$A_{ij} = \int_\omega \alpha(\mathbf{x})\langle \nabla b_i \cdot \nabla b_j \rangle d\mathbf{x} \tag{4.24}$$

For $\omega \in \mathcal{T}_h$, $F \in \mathbb{R}^{3\times 3}$ comes from the discrete integration and the matrix coefficients F_{ij} are computed using (4.28). The components of $\tilde{\mathbf{c}}$ can be computed as follows:

$$\tilde{c}_i = \int_\omega f b_i d\mathbf{x} \approx \frac{f(\mathbf{x}_1)b_i(\mathbf{x}_1) + f(\mathbf{x}_2)b_i(\mathbf{x}_2) + f(\mathbf{x}_3)b_i(\mathbf{x}_3)}{3} \cdot |\omega|, \tag{4.25}$$

where $\mathbf{x}_i$, $i \in \{1,2,3\}$, are three vertices of the triangle $\omega \in T_{\mathcal{T}_h}$, $b_i(\mathbf{x}_j) = 1$ if $i = j$ and $b_i(\mathbf{x}_j) = 0$ otherwise. Rewrite (4.25) in matrix form:

$$\tilde{\mathbf{c}} = \begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \\ \tilde{c}_3 \end{pmatrix} \approx \frac{1}{3} \begin{pmatrix} b_1(\mathbf{x}_1) & b_1(\mathbf{x}_2) & b_1(\mathbf{x}_3) \\ b_2(\mathbf{x}_1) & b_2(\mathbf{x}_2) & b_2(\mathbf{x}_3) \\ b_3(\mathbf{x}_1) & b_3(\mathbf{x}_2) & b_3(\mathbf{x}_3) \end{pmatrix} \begin{pmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ f(\mathbf{x}_3) \end{pmatrix} \cdot |\omega|, \tag{4.26}$$

where $f(\mathbf{x}_i)$, $i = 1,2,3$, are the values of the right-hand side f in the vertices of ω. Let

$$F := \frac{1}{3} \begin{pmatrix} b_1(\mathbf{x}_1) & b_1(\mathbf{x}_2) & b_1(\mathbf{x}_3) \\ b_2(\mathbf{x}_1) & b_2(\mathbf{x}_2) & b_2(\mathbf{x}_3) \\ b_3(\mathbf{x}_1) & b_3(\mathbf{x}_2) & b_3(\mathbf{x}_3) \end{pmatrix}. \tag{4.27}$$

Using the definition of basic functions, we obtain

$$\begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \\ \tilde{c}_3 \end{pmatrix} \approx \frac{1}{3} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ f(\mathbf{x}_3) \end{pmatrix} \cdot |\omega|. \tag{4.28}$$

Thus, Ψ_ω^g corresponds to the matrix $A \in \mathbb{R}^{3\times 3}$ and Ψ_ω^f to $F \in \mathbb{R}^{3\times 3}$.

4.3.2 The Recursion

The coefficients of Ψ_ω can be computed by (4.15). Let $\omega \in T_{\mathcal{T}_h}$ and ω_1, ω_2 be two sons of ω. The external boundary Γ_ω of ω splits into (see Fig. 4.2)

$$\Gamma_{\omega,1} := \partial\omega \cap \omega_1, \quad \Gamma_{\omega,2} := \partial\omega \cap \omega_2. \tag{4.29}$$

For simplicity of further notation, we will write γ instead of γ_ω.

Notation 4.3.1 *Recall that $I(\partial\omega_i) = I(\Gamma_{\omega,i}) \cup I(\gamma)$. We denote the restriction of $\Psi_{\omega_i} : \mathbb{R}^{I(\partial\omega_i)} \to \mathbb{R}^{I(\partial\omega_i)}$ to $I(\gamma)$ by ${}^\gamma\Psi_\omega := (\Psi_\omega)|_{i\in I(\gamma)}$.*

Suppose that by induction, the mappings Ψ_{ω_1}, Ψ_{ω_2} are known for the sons ω_1, ω_2. Now, we explain how to construct Ψ_ω and Φ_ω.

Lemma 4.3.1 *Let the data $d_1 = d_{\omega_1}$, $d_2 = d_{\omega_2}$ be given by (4.6).*
d_1 and d_2 coincide w.r.t. γ_ω, i.e.,
- *(consistency conditions for the boundary)*

$$g_{1,i} = g_{2,i} \quad \forall i \in I(\omega_1) \cap I(\omega_2), \tag{4.30}$$

- *(consistency conditions for the right-hand side)*

$$f_{1,i} = f_{2,i} \quad \forall i \in I(\omega_1) \cap I(\omega_2). \tag{4.31}$$

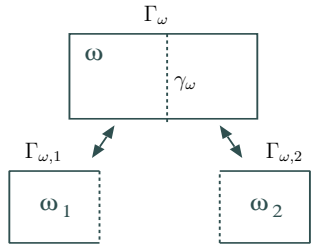


Figure 4.2: The decomposition of ω into ω_1 and ω_2. Here γ_ω is the internal boundary and $\Gamma_{\omega,i}$, $i = 1, 2$, parts of the external boundaries, see (4.29).

If the local FE solutions $u_{h,1}$ and $u_{h,2}$ of the problem (4.7) for the data d_1, d_2 satisfy the additional equation

$$ {}^{\gamma}\Psi_{\omega_1}(d_1) + {}^{\gamma}\Psi_{\omega_2}(d_2) = 0, \tag{4.32}$$

then the composed solution u_h defined by assembling

$$u_h(\mathbf{x}_i) = \begin{cases} u_{h,1}(\mathbf{x}_i) & for \quad i \in I(\omega_1), \\ u_{h,2}(\mathbf{x}_i) & for \quad i \in I(\omega_2) \end{cases} \tag{4.33}$$

satisfies (4.7) for the data $d_\omega = (f, g)$ where

$$f_i = \begin{cases} f_{1,i} & for \quad i \in I(\omega_1), \\ f_{2,i} & for \quad i \in I(\omega_2), \end{cases} \tag{4.34}$$

$$g_i = \begin{cases} g_{1,i} & for \quad i \in I(\Gamma_{\omega,1}), \\ g_{2,i} & for \quad i \in I(\Gamma_{\omega,2}). \end{cases} \tag{4.35}$$

Proof: Note that the index sets in (4.33)-(4.35) overlap. Let $\omega_1 \in T_{\mathcal{T}_h}$, $f_{1,i} = f_i$, $i \in I(\omega_1)$, and $g_{1,i} = g_i$, $i \in I(\partial\omega_1)$. Then the existence of the unique solutions of (4.7) gives $u_{h,1}(\mathbf{x}_i) = u_h(\mathbf{x}_i)$, $\forall i \in I(\mathring{\omega_1})$.
In a similar manner we get $u_{h,2}(\mathbf{x}_i) = u_h(\mathbf{x}_i)$, $\forall i \in I(\mathring{\omega_2})$. Equation (4.15) gives

$$({}^{\gamma}\Psi_{\omega_1}(d_1))_{i\in I(\gamma)} = a_{\omega_1}(u_h, b_i) - (f_{\omega_1}, b_i)_{L^2(\omega_1)} \tag{4.36}$$

and

$$({}^{\gamma}\Psi_{\omega_2}(d_2))_{i\in I(\gamma)} = a_{\omega_2}(u_h, b_i) - (f_{\omega_2}, b_i)_{L^2(\omega_2)}\,. \tag{4.37}$$

The sum of the two last equations (see Figure 4.3) and (4.32) give

$$0 = {}^{\gamma}\Psi_{\omega}(d_\omega)_{i\in I(\gamma)} = a_\omega(u_h, b_i) - (f_\omega, b_i)_{L^2(\omega)}. \tag{4.38}$$

We see that u_h satisfies (4.7). ■
Note that

$$u_{h,1}(\mathbf{x}_i) = g_{1,i} = g_{2,i} = u_{h,2}(\mathbf{x}_i) \quad \text{holds for} \quad i \in I(\omega_1) \cap I(\omega_2).$$

Next, we use the decomposition of the data d_1 into the components

$$d_1 = (f_1, g_{1,\Gamma}, g_{1,\gamma}), \tag{4.39}$$

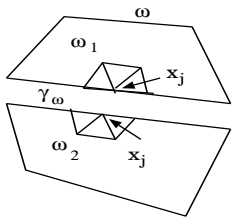


Figure 4.3: The supports of b_j, $x_j \in \omega_1$ and $x_j \in \omega_2$.

where

$$g_{1,\Gamma} := (g_1)_{i\in I(\Gamma_{\omega,1})}, \quad g_{1,\gamma} := (g_1)_{i\in I(\gamma)} \tag{4.40}$$

and similarly for $d_2 = (f_2, g_{2,\Gamma}, g_{2,\gamma})$.
The decomposition $g \in \mathbb{R}^{I(\partial\omega_j)}$ into $g_{j,\Gamma} \in \mathbb{R}^{I(\Gamma_{\omega,j})}$ and $g_{j,\gamma} \in \mathbb{R}^{I(\gamma)}$ implies the decomposition of $\Psi^g_{\omega_j} : \mathbb{R}^{I(\partial\omega_j)} \to \mathbb{R}^{I(\partial\omega_j)}$ into $\Psi^\Gamma_{\omega_j} : \mathbb{R}^{I(\Gamma_{\omega,j})} \to \mathbb{R}^{I(\partial\omega_j)}$ and $\Psi^\gamma_{\omega_j} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\partial\omega_j)}$, $j = 1, 2$. Thus,

$$\Psi^g_{\omega_1} g_{\omega_1} = \Psi^\Gamma_{\omega_1} g_{1,\Gamma} + \Psi^\gamma_{\omega_1} g_{1,\gamma} \quad \text{and} \quad \Psi^g_{\omega_2} g_{\omega_2} = \Psi^\Gamma_{\omega_2} g_{2,\Gamma} + \Psi^\gamma_{\omega_2} g_{2,\gamma}.$$

The maps Ψ_{ω_1}, Ψ_{ω_2} become

$$\Psi_{\omega_1} d_1 = \Psi^f_{\omega_1} f_1 + \Psi^\Gamma_{\omega_1} g_{1,\Gamma} + \Psi^\gamma_{\omega_1} g_{1,\gamma}, \tag{4.41}$$

$$\Psi_{\omega_2} d_2 = \Psi^f_{\omega_2} f_2 + \Psi^\Gamma_{\omega_2} g_{2,\Gamma} + \Psi^\gamma_{\omega_2} g_{2,\gamma}. \tag{4.42}$$

Definition 4.3.1 *We will denote the restriction of $\Psi^\gamma_{\omega_j} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\partial\omega_j)}$ to $I(\gamma)$ by*

$${}^\gamma\Psi^\gamma_{\omega_j} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\gamma)},$$

where $j = 1, 2$ and $\partial\omega_j = \Gamma_{\omega,j} \cup \gamma$.

Restricting (4.41), (4.42) to $I(\gamma)$, we obtain from (4.32) and $g_{1,\gamma} = g_{2,\gamma} =: g_\gamma$ that

$$\left({}^\gamma\Psi^\gamma_{\omega_1} + {}^\gamma\Psi^\gamma_{\omega_2} \right) g_\gamma = (-\Psi^f_{\omega_1} f_1 - \Psi^\Gamma_{\omega_1} g_{1,\Gamma} - \Psi^f_{\omega_2} f_2 - \Psi^\Gamma_{\omega_2} g_{2,\Gamma})|_{I(\gamma)}.$$

Next, we set $M := -({}^\gamma\Psi^\gamma_{\omega_1} + {}^\gamma\Psi^\gamma_{\omega_2})$, and after computing M^{-1}, we obtain:

$$g_\gamma = M^{-1}(\Psi^f_{\omega_1} f_1 + \Psi^\Gamma_{\omega_1} g_{1,\Gamma} + \Psi^f_{\omega_2} f_2 + \Psi^\Gamma_{\omega_2} g_{2,\Gamma})|_{I(\gamma)}. \tag{4.43}$$

Remark 4.3.1 *The inverse matrix M^{-1} exists since it is the sum of positive definite matrices corresponding to the mappings ${}^\gamma\Psi^\gamma_{\omega_1}$, ${}^\gamma\Psi^\gamma_{\omega_2}$.*

Remark 4.3.2 *Since $g_{\gamma,i} = u_h(\mathbf{x}_i)$, $i \in I(\gamma)$, we have determined the map Φ_ω (it acts on the data d_ω composed by f_1, f_2, $g_{1,\Gamma}, g_{2,\Gamma}$).*

Remark 4.3.3 *We have the formula $\Psi_\omega(d_\omega) = \Psi_{\omega_1}(d_1) + \Psi_{\omega_2}(d_2)$, where*

$$\begin{aligned} d_\omega = (f_\omega, g_\omega), \quad d_1 = (f_1, g_{1,\Gamma}, g_{1,\gamma}), \quad d_2 = (f_2, g_{2,\Gamma}, g_{2,\gamma}), \\ g_{1,\gamma} = g_{2,\gamma} = M^{-1}(\Psi^f_{\omega_1} f_1 + \Psi^\Gamma_{\omega_1} g_{1,\Gamma} + \Psi^f_{\omega_2} f_2 + \Psi^\Gamma_{\omega_2} g_{2,\Gamma})|_{I(\gamma)}. \end{aligned} \tag{4.44}$$

Here (f_ω, g_ω) is build as in (4.34)-(4.35) and (4.30),(4.31) are satisfied.

Conclusion:
Thus, using the given mappings Ψ_{ω_1}, Ψ_{ω_2}, defined on the sons $\omega_1, \omega_2 \in T_{\mathcal{T}_h}$, we can compute Φ_ω and Ψ_ω for the father $\omega \in T_{\mathcal{T}_h}$. This recursion process terminates as soon as $\omega = \Omega$.

4.3.3 Building of Matrices Ψ_ω and Φ_ω from Ψ_{ω_1} and Ψ_{ω_2}

Let ω, ω_1 and $\omega_2 \in T_{\mathcal{T}_h}$ where ω_1, ω_2 are sons of ω. Recall that $\partial\omega_i = \Gamma_{\omega,i} \cup \gamma$. Suppose we have two linear systems of equations for ω_1 and ω_2 which can be written in the block-matrix form:

$$\begin{pmatrix} A_{11}^{(i)} & A_{12}^{(i)} \\ A_{21}^{(i)} & A_{22}^{(i)} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1^{(i)} \\ \mathbf{u}_2^{(i)} \end{pmatrix} = \begin{pmatrix} F_{11}^{(i)} & F_{12}^{(i)} \\ F_{21}^{(i)} & F_{22}^{(i)} \end{pmatrix} \begin{pmatrix} \mathbf{c}_1^{(i)} \\ \mathbf{c}_2^{(i)} \end{pmatrix}, \quad i = 1, 2, \tag{4.45}$$

where $\gamma := \gamma_\omega$,

$$A_{11}^{(i)} : \mathbb{R}^{I(\Gamma_{\omega,i})} \to \mathbb{R}^{I(\Gamma_{\omega,i})}, \quad A_{12}^{(i)} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\Gamma_{\omega,i})},$$

$$A_{21}^{(i)} : \mathbb{R}^{I(\Gamma_{\omega,i})} \to \mathbb{R}^{I(\gamma)}, \quad A_{22}^{(i)} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\gamma)},$$

$$F_{11}^{(i)} : \mathbb{R}^{I(\omega_i \setminus \gamma)} \to \mathbb{R}^{I(\partial\omega_i)}, \quad F_{12}^{(i)} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\partial\omega_i)},$$

$$F_{21}^{(i)} : \mathbb{R}^{I(\omega_i \setminus \gamma)} \to \mathbb{R}^{I(\gamma)}, \quad F_{22}^{(i)} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\gamma)}.$$

Both the equations in (4.45) are analogous to (4.41) and (4.42). Note that $\mathbf{c}_2^{(1)} = \mathbf{c}_2^{(2)}$ and $\mathbf{u}_2^{(1)} = \mathbf{u}_2^{(2)}$ because of the consistency conditions (see (4.30),(4.31)) on the interface γ. The system of linear equations for ω be

$$\begin{pmatrix} A_{11}^{(1)} & 0 & A_{12}^{(1)} \\ 0 & A_{11}^{(2)} & A_{12}^{(2)} \\ A_{21}^{(1)} & A_{21}^{(2)} & A_{22}^{(1)} + A_{22}^{(2)} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1^{(1)} \\ \mathbf{u}_1^{(2)} \\ \mathbf{u}_2^{(1)} \end{pmatrix} = \begin{pmatrix} F_{11}^{(1)} & 0 & F_{12}^{(1)} \\ 0 & F_{11}^{(2)} & F_{12}^{(2)} \\ F_{21}^{(1)} & F_{21}^{(2)} & F_{22}^{(1)} + F_{22}^{(2)} \end{pmatrix} \begin{pmatrix} \mathbf{c}_1^{(1)} \\ \mathbf{c}_1^{(2)} \\ \mathbf{c}_2^{(1)} \end{pmatrix}. \tag{4.46}$$

Using the notation

$$\tilde{A}_{11} := \begin{pmatrix} A_{11}^{(1)} & 0 \\ 0 & A_{11}^{(2)} \end{pmatrix}, \quad \tilde{A}_{12} := \begin{pmatrix} A_{12}^{(1)} \\ A_{12}^{(2)} \end{pmatrix},$$

$$\tilde{A}_{21} := (A_{21}^{(1)}, A_{21}^{(2)}), \quad \tilde{A}_{22} := A_{22}^{(1)} + A_{22}^{(2)},$$

$$\tilde{\mathbf{u}}_1 := \begin{pmatrix} \mathbf{u}_1^{(1)} \\ \mathbf{u}_1^{(2)} \end{pmatrix}, \quad \tilde{\mathbf{u}}_2 := \mathbf{u}_2^{(1)} = \mathbf{u}_2^{(2)},$$

$$\tilde{F}_1 := \begin{pmatrix} F_{11}^{(1)} & 0 & F_{12}^{(1)} \\ 0 & F_{11}^{(2)} & F_{12}^{(2)} \end{pmatrix}, \quad \tilde{F}_2 := \begin{pmatrix} F_{21}^{(1)}, & F_{21}^{(2)}, & F_{22}^{(1)} + F_{22}^{(2)} \end{pmatrix},$$

$$\tilde{\mathbf{c}}_1 := \begin{pmatrix} \mathbf{c}_1^{(1)} \\ \mathbf{c}_1^{(2)} \end{pmatrix}, \quad \tilde{\mathbf{c}}_2 := \mathbf{c}_2^{(1)} = \mathbf{c}_2^{(2)}, \quad \tilde{\mathbf{c}} := \begin{pmatrix} \tilde{\mathbf{c}}_1 \\ \tilde{\mathbf{c}}_2 \end{pmatrix}.$$

The system (4.46) can be rewritten as

$$\begin{pmatrix} \tilde{A}_{11}^{(i)} & \tilde{A}_{12}^{(i)} \\ \tilde{A}_{21}^{(i)} & \tilde{A}_{22}^{(i)} \end{pmatrix} \begin{pmatrix} \tilde{\mathbf{u}}_1 \\ \tilde{\mathbf{u}}_2 \end{pmatrix} = \begin{pmatrix} \tilde{F}_1 \\ \tilde{F}_2 \end{pmatrix} \tilde{\mathbf{c}}. \tag{4.47}$$

The system (4.47), indeed, coincides with (4.16).

4.3.4 Algorithm "Leaves to Root"

The scheme of the recursive process of computing Ψ_ω and Φ_ω from Ψ_{ω_1} and Ψ_{ω_2} for all $\omega \in T_{\mathcal{T}_h}$ is shown in Fig. 4.4. We call this process "**Leaves to Root**".

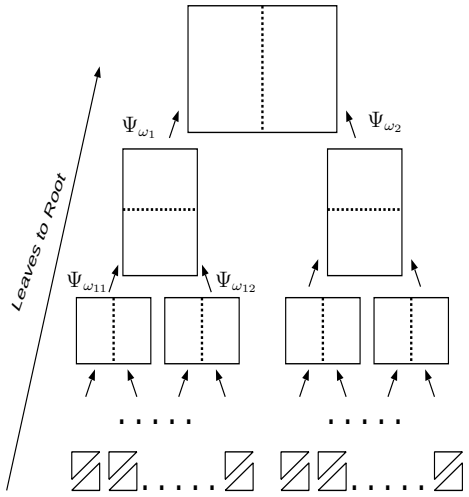


Figure 4.4: Recursive process "Leaves to Root". A mapping Ψ_{ω_1} is a linear function of the mappings $\Psi_{\omega_{11}}, \Psi_{\omega_{12}}$, $\omega_1 = \omega_{11} \cup \omega_{12}$.

"**Leaves to Root**":

1. Compute $\Psi_\omega^f \in \mathbb{R}^{3\times 3}$ and $\Psi_\omega^g \in \mathbb{R}^{3\times 3}$ on all leaves of $T_{\mathcal{T}_h}$ (triangles of $\mathcal{T}_h$) by (4.24) and (4.28).

2. Recursion from the leaves to the root:
   a) Compute Φ_ω and Ψ_ω from $\Psi_{\omega_1}, \Psi_{\omega_2}$.
   b) Store Φ_ω. Delete $\Psi_{\omega_1}, \Psi_{\omega_2}$.

3. Stop if $\omega = \Omega$.

Remark 4.3.4 *The result of this algorithm will be a collection of mappings* $\{\Phi_\omega : \omega \in T_{\mathcal{T}_h}\}$. *The mappings* Ψ_ω, $\omega \in T_{\mathcal{T}_h}$, *are only of auxiliary purpose and need not stored.*

4.3.5 Algorithm "Root to Leaves"

This algorithm applies the mappings $\Phi_\omega = (\Phi_\omega^g, \Phi_\omega^f)$ to compute the solution. Since this algorithm starts from the root and ends on the leaves, we will use the name "**Root to Leaves**". Figure 4.5 presents the scheme of this algorithm.
Let the set $\{\Phi_\omega : \omega \in T_{\mathcal{T}_h}\}$ already be computed and the data $d_\omega = (f_\omega, g_\omega)$, $\omega = \Omega$, be given. We can then compute the solution u_h of the initial problem as follows.
Recursion from the root to the leaves:

1. Given $d_\omega = (f_\omega, g_\omega)$, compute the solution u_h on the interior boundary γ_ω by $\Phi_\omega(d_\omega)$.

2. Build the data $d_{\omega_1} = (f_{\omega_1}, g_{\omega_1})$, $d_{\omega_2} = (f_{\omega_2}, g_{\omega_2})$ from $d_\omega = (f_\omega, g_\omega)$ and $g_{\gamma_\omega} := \Phi_\omega(d_\omega)$.

3. Repeat the same for the sons of ω_1 and ω_2.

4. End if ω does not contain internal nodes.

Since $u_h(\mathbf{x}_i) = g_{\gamma,i}$, the set of values (g_{γ_ω}), for all $\omega \in T_{\mathcal{T}_h}$, results the solution of the initial problem (4.1) in the whole domain Ω.

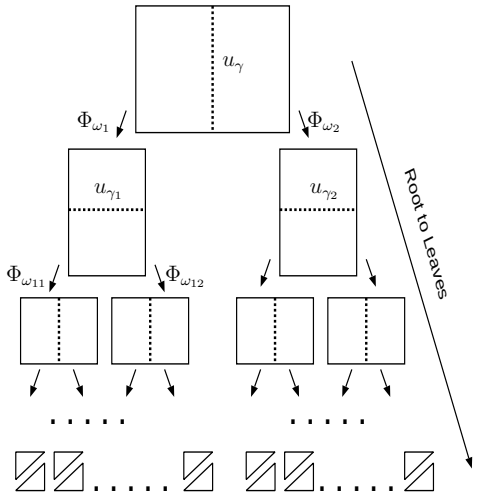


Figure 4.5: The algorithm 'Root to Leaves'. Φ_{ω_i} is applied for computing the solution on the interior boundary γ_i.

4.3.6 HDD on Two Grids

In (4.4) the operator $\mathcal{F}_h$ requires larger computational resources compared to $\mathcal{G}_h$. To reduce these requirements we compute u_h in the following way

$$u_h(f_H, g_h) = \mathcal{F}_H f_H + \mathcal{G}_h g_h,$$

where $F_H := \mathcal{F}_h P_{h \leftarrow H}$, H and h are two different scales and $P_{h \leftarrow H}$ is a prolongation matrix.
Now, F_H requires less resources (computational time and storage requirement) than $\mathcal{F}_h$. If the coefficients $\alpha(\mathbf{x})$ oscillate, the grid step size should be small enough to catch these oscillations (denote this step size by "h" and corresponding finite space by V_h). At the same time the right-hand side $f(\mathbf{x})$ is often smooth and therefore can be discretised on a coarse grid with step size H, where $H > h$ (see Fig. 4.6). Another possible problem setup is when the right-hand side is only given at the nodes of the coarse grid.

The algorithm of computing the prolongation matrix $P := P_{h \leftarrow H}$

Note that we compute prolongation matrices only for triangles of $\mathcal{T}_H$. For simplicity we write P instead of $P_{h \leftarrow H}$. Let $h = \frac{H}{2}$. To compute $f_h = P f_H$ we perform the following steps.

1. In each vertex v_j, $j = 0, 1, 2$, of $\omega \in \mathcal{T}_H$ we compute the piecewise linear basis function $b_j^H(\mathbf{x})$ by (3.29).

2. We compute the prolongation matrix $P \in \mathbb{R}^{3\times 3}$, where $P_{ij} = b_j^H(\mathbf{x}_i)$, $\mathbf{x}_i$ are nodes of the fine grid $\mathcal{T}_h$ in ω (e.g., middle points of edges). The value of the right-hand side f at a node of $\mathcal{T}_h$ is defined as a linear combination of the values f at the nodes of $\mathcal{T}_H$ (see an example in Fig. 4.7).

Remark 4.3.5 *Now, we compute the solution by (cf. 4.43):*

$$g_\gamma = M^{-1}(\tilde{\Psi}^f_{\omega_1} f_1 + \Psi^\Gamma_{\omega_1} g_{1,\Gamma} + \tilde{\Psi}^f_{\omega_2} f_2 + \Psi^\Gamma_{\omega_2} g_{2,\Gamma})|_{I(\gamma)}, \tag{4.48}$$

where $\tilde{\Psi}^f_{\omega_i} := \Psi^f_{\omega_i} P_i$, $i = 1, 2$, *and* P_i *is a prolongation matrix. The size of the matrix* $\tilde{\Psi}^f_{\omega_i}$ *is smaller than the size of the matrix* $\Psi^f_{\omega_i}$, $i = 1, 2$. *The size of the matrix* $\Phi^f_{\omega_i} P_i$ *is smaller than the size of* $\Phi^f_{\omega_i}$, $i = 1, 2$. *The matrices* P_1 *and* P_2 *are data-sparse, since basis functions have a local support. Thus, memory requirements can be strongly diminished.*

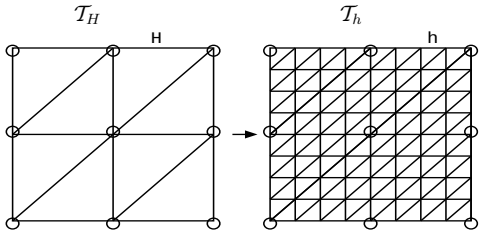


Figure 4.6: The right-hand side f is given at the grid points of $\mathcal{T}_H$ (marked by ($\circ$)) and has to be interpolated at the nodes of the fine grid $\mathcal{T}_h$ (right).

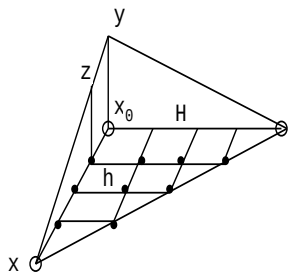


Figure 4.7: $\omega \in \mathcal{T}_H$ and a basis function at $\mathbf{x}_0$. For example, the value of f at the point $\mathbf{z}$ a linear combination on the values of f at the vertices $\mathbf{x}, \mathbf{y}$ of ω (e.g., $f(\mathbf{z}) = 0.75f(\mathbf{y}) + 0.25f(\mathbf{x})$).

4.4 Modifications of HDD

4.4.1 Truncation of Small Scales

The standard version of HDD deals with one grid with step size h. In this subsection we consider HDD on two grids with step sizes h and H, $h < H$. Often, only the solution on the coarse scale is of interest, but to build this solution accurately one should take into account the fine scale details.

Notation 4.4.1 *Let us denote the pruned domain decomposition tree by* $T_{\mathcal{T}_h}^{\geq H}$ *and the part removed by pruning by* $T_{\mathcal{T}_h}^{<H}$ *(for this part* $diam(\omega) \leq H$*). Thus, we have* $T_{\mathcal{T}_h} = T_{\mathcal{T}_h}^{<H} \cup T_{\mathcal{T}_h}^{\geq H}$ *(see Fig. 4.8).*

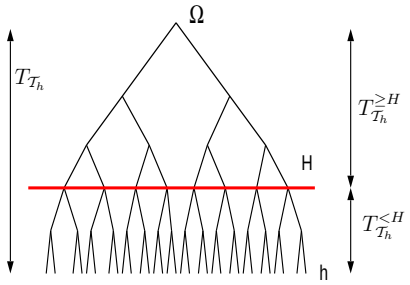


Figure 4.8: Truncation of the small scales.

Remark 4.4.1 *The algorithm "Root to Leaves" determines* u_h *in all* $\omega \in T_{\mathcal{T}_h}^{\geq H}$ *and terminates on the level corresponding to a medium scale* H *(see Fig. 4.8). The mappings* Φ_ω, $\omega \in T_{\mathcal{T}_h}^{<H}$, *are not computed and not stored. This truncation of the small scales does not imply any additional error to the solution in* $\omega \in T_{\mathcal{T}_h}^{\geq H}$.

Remark 4.4.2 *In Chapter 10, we compute the complexities of the algorithms "Leaves to Root", "Root to Leaves" and their modifications.*

Now, we compare our method with the multiscale finite element method (MsFEM) (see [50], [40]):

- MsFEM requires two different triangulations. The first triangulation is needed for solving the appropriate homogenized problem inside a cell and the second one for solving the macroscale problem. HDD requires one coarse triangulation and its refinement.
- The solution which MsFEM computes on the coarse scale with step size H does not coincide with the solution which the standard FEM with step size h would produce. In MsFEM the accuracy of the solution on the coarse scale depends on the accuracy of the solver in the cell (different from HDD). The solution, produced by HDD with truncation of the small scales, does not contain any additional error.

- In MsFEM it is not obvious how to build the boundary data for the homogenized problem in 2D and 3D cases in a non-periodic case.
- The offered version of HDD is applicable only in 2D. In the present time the 3D version of HDD is under construction.
- MsFEM requires periodicity (unlike HDD) and as a sequence the complexity of MsFEM is $\mathcal{O}(n_H+n_s)$, where n_H is the number of cells (equal to the number of degrees of freedom on the coarse grid) and n_s the number of degrees of freedom in the cell. HDD with repeated patterns (see Section 4.4.4) has complexity $\mathcal{O}(n_H \log^3 n_H) + \mathcal{O}(n_s \log^3 n_s)$. Note that MsFEM applies multigrid method and computes a particular solution, whereas HDD computes the mappings, i.e., HDD is appropriate for the many right-hand sides.
- HDD can compute different functionals of the solution.
- HDD has linear complexity for homogeneous problems (Section 4.4.8).

Both HDD and MsFEM have the advantage that they are easy parallelizable.

4.4.2 Two-Grid Modification of the Algorithm "Leaves to Root"

The purposes of this modification are:

1. to decrease the memory requirements of the matrices Ψ^f_ω and Φ^f_ω,
2. to speed up the construction of the matrix Ψ^f_ω from the matrices $\Psi^f_{\omega_1}$ and $\Psi^f_{\omega_2}$, where $\omega = \omega_1 \cup \omega_2$.

Let us introduce some **new notation**:

1. The subindex $_h$ indicates a fine grid.
2. The subindex $_H$ indicates a coarse grid.
3. The numbers of grid points in ω are denoted by $n_h(\omega) := |I(\omega_h)|$ and $n_H(\omega) := |I(\omega_H)|$ respectively.
4. The index sets of indices in ω_i are denoted by $I(\omega_{i,h})$ and $I(\omega_{i,H})$ respectively.
5. $P_i : \mathbb{R}^{I(\omega_{i,H})} \to \mathbb{R}^{I(\omega_{i,h})}$, $i = 1, 2$, are the prolongation matrices.

Compression of the Mapping Ψ^f_ω

The algorithm "Leaves to Root" computes the mappings:

$$\Psi^f_\omega : \mathbb{R}^{I(\omega_h)} \to \mathbb{R}^{I(\partial\omega_h)}, \quad \text{for all } \omega \in T_{\mathcal{T}_h}.$$

Taking into account the results from Section 4.3.6, the modified version of the algorithm computes the new mappings:

$$\tilde{\Psi}^f_\omega : \mathbb{R}^{I(\omega_H)} \to \mathbb{R}^{I(\partial\omega_h)}, \text{ for all } \omega \in T_{\mathcal{T}_h}.$$

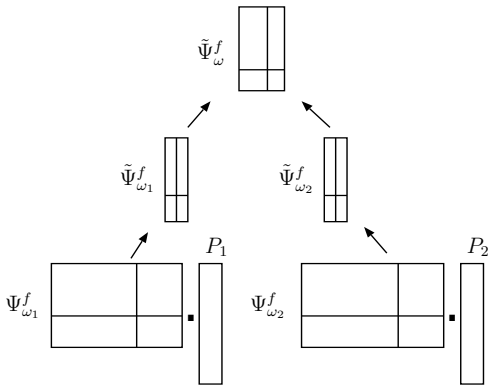


Figure 4.9: The matrix $\tilde{\Psi}^f_\omega$ is constructed from the matrices $\tilde{\Psi}^f_{\omega_1}$ and $\tilde{\Psi}^f_{\omega_2}$ (one step of the algorithm "Leaves to Root").

The new scheme of the construction of $\tilde{\Psi}^f_\omega$ from $\tilde{\Psi}^f_{\omega_1}$ and $\tilde{\Psi}^f_{\omega_2}$ is shown in Fig. 4.9. The number of columns in $\tilde{\Psi}^f_\omega$ is by a factor $\frac{n_H(\omega)}{n_h(\omega)}$ smaller than the number of columns in Ψ^f_ω.

Remark 4.4.3 *In practice, we construct prolongation matrices only for leaves of $T_{\mathcal{T}_h}$. Then we construct $\tilde{\Psi}^f_\omega$ directly from $\tilde{\Psi}^f_{\omega_1}$ and $\tilde{\Psi}^f_{\omega_2}$.*

Compression of the mapping Φ^f_ω

The algorithm "Leaves to Root" computes also the mappings:

$$\Phi^f_\omega : \mathbb{R}^{I(\omega_h)} \to \mathbb{R}^{I(\gamma_h)}, \quad \text{for all } \omega \in T_{\mathcal{T}_h}.$$

Taking into account the results from Section (4.3.6), we compute the new mappings:

$$\tilde{\Phi}^f_\omega : \mathbb{R}^{I(\omega_H)} \to \mathbb{R}^{I(\gamma_h)}, \quad \text{for all } \omega \in T_{\mathcal{T}_h}.$$

The idea of the compression is shown in Fig. 4.10. There an additional prolongation matrix P is applied. As a result the number of columns in the new matrix $\tilde{\Phi}^f_\omega$ is in a factor $\frac{n_H(\omega)}{n_h(\omega)}$ smaller than in Φ^f_ω.

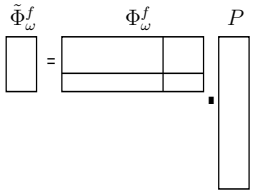


Figure 4.10: Compression of the matrix Φ^f_ω.

4.4.3 HDD on two grids and with Truncation of Small Scales

An application of HDD on two grids with truncation of the small scales results in the optimal performance. The realization of the HDD method stays the same with the difference that the domain of definition of both mappings Ψ^f_ω and Φ^f_ω is $\mathbb{R}^{I(\omega_H)}$, instead of $\mathbb{R}^{I(\omega_h)}$.

4.4.4 Repeated Patterns

Elliptic partial differential equations with periodically oscillating coefficients were considered in [23, 52, 53, 42]. For solving such equations the authors used the theory of homogenization and different modifications of multigrid methods.
The domain with repeated patterns can be decomposed into subdomains (we call them *cells*), where the computations are equivalent up to translation. In this case, only one cell has to be considered in the computation (see Fig. 4.12 (a)). This reduces the work required in the part $T_{\mathcal{T}_h}^{\leq H}$ (see Fig. 4.8) significantly. For simplicity, suppose that the decomposition into cells forms a coarse scale with the grid step size H.
The modified HDD method is as follows:

1. Build a triangulation of the domain Ω so that the repeated cells have an identical triangulation.

2. Build the hierarchical domain decomposition tree $T_{\mathcal{T}_h}^{\geq H}$ (for each subdomain its internal and external boundary nodes must be known). The leaves of this tree are cells.

3. Build the hierarchical domain decomposition tree $T_v := T_{\mathcal{T}_h}(v)$ just for one cell v.

4. Run algorithm "Leaves to Root" for the tree T_v (see Fig. 4.12(a)). This algorithm computes the mappings Φ_ω^f and Φ_ω^g for all $\omega \in T_v$.

5. If the symmetry allows, we compute Ψ_ω and Φ_ω on each level l of $T_{\mathcal{T}_h}^{\geq H}$ only for one node and then copy them to other nodes at the same level (see Fig. 4.12(b)).

6. Taking into account translation of indices, "Root to Leaves " computes the solution in all $\omega \in T_{\mathcal{T}_h}$.

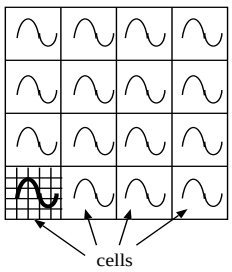


Figure 4.11: The given domain is decomposed into m cells and then each cell is decomposed independently into n_c finite elements. The oscillatory coefficients inside cells are equal.

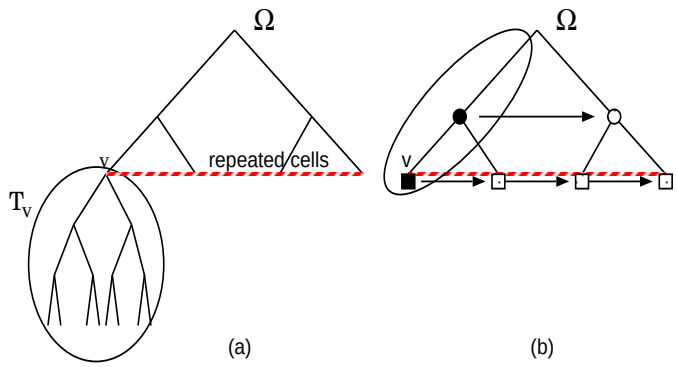


Figure 4.12: HDD with repeated patterns. (a) Φ^f_ω and Φ^g_ω, $\omega \in T_v$, are computed only for the tree T_v; (b) Φ^f_ω and Φ^g_ω, $\omega \in T^{\geq H}_{\mathcal{T}_h}$, are computed only once on each level and then copied to other nodes.

4.4.5 Fast Evaluation of Functionals

In this subsection we describe how to use the mappings Φ^f_ω and Φ^g_ω for building different linear functionals of the solution (see examples below). Indeed, λ is determined in the same way as Ψ_ω.
If the solution u in a subdomain $\omega \in T_{\mathcal{T}_h}$ is known, the mean value $\mu(\omega)$ can be computed by the following formula

$$\mu(\omega) = \frac{\int_\omega u(\mathbf{x})d\mathbf{x}}{|\omega|} = \frac{\sum_{t\in\mathcal{T}_h(\omega)} \frac{|t|}{3}(u_1+u_2+u_3)}{|\omega|}, \tag{4.49}$$

where u is affine on each triangle t with values u_1, u_2, u_3 at the three corners and $\mathcal{T}_h(\omega)$ is the collection of all triangles in ω. If the solution u is unknown, we would like to have a linear functional $\lambda_\omega(f,g), \omega \in T_{\mathcal{T}_h}$, which computes the mean value μ_ω of the solution in ω.
Below we list some examples of problems which can be solved by using linear functionals.

Example 4.4.1 *Dirichlet data on $\partial\omega$, $\omega \subset \Omega$, are given and one needs to evaluate the Neumann data $\frac{\partial u_h}{\partial n}$ on $\partial\omega$.*

Example 4.4.2 *The given domain Ω is decomposed into p subdomains $\Omega = \bigcup_{i=1}^p \Omega_i$ (see (4.2)). We denote the set of nodal points on the interface by I_Σ. The computation of the solution in the whole domain Ω can be expensive (or even impossible in a reasonable time) and, therefore, as an alternative, HDD offers the solution on the interface I_Σ and the mean values inside Ω_i, $i = 1, ..., p$.*

Example 4.4.3 *To compute the FE solution $u_h(\mathbf{x}_i)$ in a fixed nodal point $\mathbf{x}_i \in \Omega$, i.e., to define how the solution $u_h(\mathbf{x}_i)$ depends on the given FE Dirichlet data $g_h \in \mathbb{R}^{I(\partial\Omega)}$ and the FE right-hand side $f_h \in \mathbb{R}^{I(\Omega)}$.*

Let $\omega = \omega_1 \cup \omega_2$, $\omega_1 \cap \omega_2 \neq \varnothing$, with ω, ω_1, $\omega_2 \in T_{\mathcal{T}_h}$. To simplify the notation we will write $d_i := d_{\omega_i}$ and (f_i, g_i) instead of $(f_{\omega_i}, g_{\omega_i})$, $i = 1, 2$. Recall the following

notation (see (4.39), (4.40)):

$$\Gamma := \partial\omega, \quad \Gamma_{\omega,1} := \partial\omega \cap \omega_1, \quad \Gamma_{\omega,2} := \partial\omega \cap \omega_2, \quad \text{then}$$

$$d_1 = (f_1, g_1) = (f_1, g_{1,\Gamma}, g_{1,\gamma}), \quad d_2 = (f_2, g_2) = (f_2, g_{2,\Gamma}, g_{2,\gamma}), \text{ where} \tag{4.50}$$

$$\begin{array}{ll} g_{1,\Gamma} = (g_1)|_{\Gamma_{\omega,1}}, & g_{1,\gamma} = (g_1)|_{\gamma}, \\ g_{2,\Gamma} = (g_2)|_{\Gamma_{\omega,2}}, & g_{2,\gamma} = (g_2)|_{\gamma}. \end{array} \tag{4.51}$$

We consider a linear functional λ_ω with the properties:

$$\lambda_\omega(d_\omega) = (\lambda^g_\omega, g_\omega) + (\lambda^f_\omega, f_\omega), \tag{4.52}$$

$$\lambda_\omega(d_\omega) = c_1\lambda_{\omega_1}(d_{\omega_1}) + c_2\lambda_{\omega_2}(d_{\omega_2}), \tag{4.53}$$

where $\lambda^g_\omega : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}$, $\lambda^f_\omega : \mathbb{R}^{I(\omega)} \to \mathbb{R}$ and $(\cdot,\cdot)$ is the scalar product of two vectors.

Definition 4.4.1 *Let $\omega_1 \subset \omega$, $\lambda^f_{\omega_1} : \mathbb{R}^{I(\omega_1)} \to \mathbb{R}$. a) We define the following extension of $\lambda^f_{\omega_1}$*

$$(\lambda^f_{\omega_1}|^{\omega})_i := \begin{cases} (\lambda^f_{\omega_1})_i & \text{for } i \in I(\omega_1), \\ 0 & \text{for } i \in I(\omega \setminus \omega_1), \end{cases}$$

where $(\lambda^f_{\omega_1}|^{\omega}) : \mathbb{R}^{I(\omega)} \to \mathbb{R}$. b) The extension of the functional $\lambda^g_{1,\Gamma} : \mathbb{R}^{I(\Gamma_{\omega,1})} \to \mathbb{R}$ is defined as

$$(\lambda^g_{1,\Gamma}|^{\Gamma})_i := \begin{cases} (\lambda^g_{1,\Gamma})_i & \text{for } i \in I(\Gamma_{\omega,1}), \\ 0 & \text{for } i \in I(\Gamma \setminus \Gamma_{\omega,1}), \end{cases}$$

where $(\lambda^g_{1,\Gamma}|^{\Gamma}) : \mathbb{R}^{I(\Gamma)} \to \mathbb{R}$.

Definition 4.4.2 *Using (4.51), we obtain the following decompositions $\lambda^g_{\omega_1} = (\lambda^g_{1,\Gamma}, \lambda^g_{1,\gamma})$ and $\lambda^g_{\omega_2} = (\lambda^g_{2,\Gamma}, \lambda^g_{2,\gamma})$, where $\lambda^g_{1,\Gamma} : \mathbb{R}^{I(\Gamma_{\omega,1})} \to \mathbb{R}$, $\lambda^g_{1,\gamma} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}$, $\lambda^g_{2,\Gamma} : \mathbb{R}^{I(\Gamma_{\omega,2})} \to \mathbb{R}$, $\lambda^g_{2,\gamma} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}$.*

Lemma 4.4.1 *Let $\lambda_\omega(d_\omega)$ satisfy (4.52) and (4.53) with $\omega = \omega_1 \cup \omega_2$. Let $\lambda^g_{\omega_1}$, $\lambda^g_{\omega_2}$, $\lambda^f_{\omega_1}$ and $\lambda^f_{\omega_2}$ be the vectors for the representation of the functionals $\lambda_{\omega_1}(d_{\omega_1})$ and $\lambda_{\omega_2}(d_{\omega_2})$. Then the vectors λ^f_ω, λ^g_ω for the representation*

$$\lambda_\omega(d_\omega) = (\lambda^f_\omega, f_\omega) + (\lambda^g_\omega, g_\omega), \text{ where } f_\omega \in \mathbb{R}^{I(\omega)}, g_\omega \in \mathbb{R}^{I(\partial\omega)}, \tag{4.54}$$

are given by

$$\lambda^f_\omega = \tilde{\lambda}^f_\omega + (\Phi^f_\omega)^T\lambda^g_\gamma,$$

$$\lambda^g_\omega = \tilde{\lambda}^g_\omega + (\Phi^g_\omega)^T\lambda^g_\gamma,$$

$$\tilde{\lambda}^f_\omega := c_1\lambda^f_{\omega_1}|^{\omega} + c_2\lambda^f_{\omega_2}|^{\omega}, \tag{4.55}$$

$$\tilde{\lambda}^g_\Gamma := c_1\lambda^g_{1,\Gamma}|^{\Gamma} + c_2\lambda^g_{2,\Gamma}|^{\Gamma}, \tag{4.56}$$

$$\lambda^g_\gamma = c_1\lambda^g_{1,\gamma} + c_2\lambda^g_{2,\gamma}. \tag{4.57}$$

Proof: Let d_{ω_1}, d_{ω_2} be the given data and λ_{ω_1} and λ_{ω_2} be the given functionals. Then the functional λ_ω satisfies

$$\begin{aligned}\lambda_\omega(d_\omega) &\overset{(4.53)}{=} c_1\lambda_{\omega_1}(d_{\omega_1}) + c_2\lambda_{\omega_2}(d_{\omega_2}) \\ &\overset{(4.52)}{=} c_1((\lambda^f_{\omega_1}, f_1) + (\lambda^g_{\omega_1}, g_1)) + c_2((\lambda^f_{\omega_2}, f_2) + (\lambda^g_{\omega_2}, g_2)).\end{aligned}$$

Using the decomposition (4.50), we obtain

$$\begin{aligned}\lambda_\omega(d_\omega) = c_1(\lambda^f_{\omega_1}, f_1) + c_2(\lambda^f_{\omega_2}, f_2) + c_1((\lambda^g_{1,\Gamma}, g_{1,\Gamma}) + (\lambda^g_{1,\gamma}, g_{1,\gamma})) \\ +c_2((\lambda^g_{2,\Gamma}, g_{2,\Gamma}) + (\lambda^g_{2,\gamma}, g_{2,\gamma})).\end{aligned} \tag{4.58}$$

The consistency of the solution implies $g_{1,\gamma} = g_{2,\gamma} =: g_\gamma$. From the Definition 4.4.1 follows

$$\begin{aligned}(\lambda^f_{\omega_1}, f_1) = (\lambda^f_{\omega_1}|^\omega, f_\omega), \quad (\lambda^f_{\omega_2}, f_2) = (\lambda^f_{\omega_2}|^\omega, f_\omega), \\ (\lambda^g_{1,\Gamma}, g_{1,\Gamma}) = (\lambda^g_{1,\Gamma}|^\Gamma, g_\omega), \quad (\lambda^g_{2,\Gamma}, g_{2,\Gamma}) = (\lambda^g_{2,\Gamma}|^\Gamma, g_\omega).\end{aligned}$$

Then, we substitute last expressions in (4.58) to obtain

$$\begin{aligned}\lambda_\omega(d_\omega) = (c_1\lambda^f_{\omega_1}|^\omega + c_2\lambda^f_{\omega_2}|^\omega, f_\omega) + (c_1\lambda^g_{1,\Gamma}|^\Gamma + c_2\lambda^g_{2,\Gamma}|^\Gamma, g_\omega) \\ +(c_1\lambda^g_{1,\gamma} + c_2\lambda^g_{2,\gamma}, g_\gamma).\end{aligned} \tag{4.59}$$

Set $\tilde{\lambda}^f_\omega := c_1\lambda^f_{\omega_1}|^\omega + c_2\lambda^f_{\omega_2}|^\omega$, $\tilde{\lambda}^g_\Gamma := c_1\lambda^g_{1,\Gamma}|^\Gamma + c_2\lambda^g_{2,\Gamma}|^\Gamma$and $\lambda^g_\gamma := c_1\lambda^g_{1,\gamma} + c_2\lambda^g_{2,\gamma}$. From the algorithm "Root to Leaves" we know that

$$g_\gamma = \Phi_\omega(d_\omega) = \Phi^g_\omega \cdot g_\omega + \Phi^f_\omega \cdot f_\omega. \tag{4.60}$$

Substituting g_γ from (4.60) in (4.59), we obtain

$$\begin{aligned}\lambda_\omega(d_\omega) &= (\tilde{\lambda}^f_\omega, f_\omega) + (\tilde{\lambda}^g_\omega, g_\omega) + (\lambda^g_\gamma, \Phi^g_\omega g_\omega + \Phi^f_\omega f_\omega) \\ &= (\tilde{\lambda}^f_\omega + (\Phi^f_\omega)^T\lambda^g_\gamma, f_\omega) + (\tilde{\lambda}^g_\omega + (\Phi^g_\omega)^T\lambda^g_\gamma, g_\omega).\end{aligned}$$

We define $\lambda^f_\omega := \tilde{\lambda}^f_\omega + (\Phi^f_\omega)^T\lambda^g_\gamma$ and $\lambda^g_\omega := \tilde{\lambda}^g_\omega + (\Phi^g_\omega)^T\lambda^g_\gamma$ and obtain

$$\lambda_\omega(d_\omega) = (\lambda^f_\omega, f_\omega) + (\lambda^g_\omega, g_\omega). \tag{4.61}$$

■

Example 4.4.4 *Lemma 4.4.1 with* $c_1 = \frac{|\omega_1|}{|\omega|}$, $c_2 = \frac{|\omega_2|}{|\omega|}$ *can be used to compute the mean values in all* $\omega \in T_{\mathcal{T}_h}$.

4.4.6 Functional for Computing the Mean Value

Below we describe two algorithms which are required for computing mean values. These algorithms compute λ_ω^g and λ_ω^f respectively.

Algorithm
The initialisation is $\lambda_\omega^g := (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, $\lambda_\omega^f := (0,0,0)$ for all leaves of $T_{\mathcal{T}_h}$. For all $\omega \in T_{\mathcal{T}_h}$ which have internal nodes the algorithms for building λ_ω^g and λ_ω^f, $\omega \in T_{\mathcal{T}_h}$, are the following:

Algorithm 4.4.1 *(Building of λ_ω^g)*
build_functional_g$(\lambda_1^g, \lambda_2^g, \Phi_\omega^g, \ldots)$
begin
allocate memory for λ_ω^g;
for all $i \in I(\Gamma_{\omega,1})$ ***do***
$\lambda_\omega^g[i] += c_1 \lambda_1^g[i]$;
for all $i \in I(\Gamma_{\omega,2})$ ***do***
$\lambda_\omega^g[i] += c_2 \lambda_2^g[i]$;
for all $i \in I(\gamma)$ ***do***
$z[i] = c_1 \lambda_1^g[i] + c_2 \lambda_2^g[i]$;
$v := (\Phi_\omega^g)^T \cdot z$;
for all $i \in I(\partial\omega)$ ***do***
$\lambda_\omega^g[i] := \lambda_\omega^g[i] + v[i]$;
return λ_ω^g;
end;

Algorithm 4.4.2 *(Building of λ_ω^f)*
build_functional_f$(\lambda_1^f, \lambda_2^f, \Phi_\omega^f, \ldots)$
begin
for all $i \in I(\omega_1 \backslash \gamma)$ ***do***
$\lambda_\omega^f[i] += c_1 \lambda_1^f[i]$;
for all $i \in I(\omega_2 \backslash \gamma)$ ***do***
$\lambda_\omega^f[i] += c_2 \lambda_2^f[i]$;
for all $i \in I(\gamma)$ ***do***
$z[i] = c_1 \lambda_1^f[i] + c_2 \lambda_2^f[i]$;
$v := (\Phi_\omega^f)^T \cdot z$;
for all $i \in I(\omega)$ ***do***
$\lambda_\omega^f[i] := \lambda_\omega^f[i] + v[i]$;
return λ_ω^f;
end;

Remark 4.4.4 *a) If only the functionals λ_ω, $\omega \in T_{\mathcal{T}_h}$, are of interest, the maps Φ_ω need not be stored.*
b) For functionals with local support in some $\omega_0 \in T_{\mathcal{T}_h}$, it suffices that Φ_ω is given for all $\omega \in T_{\mathcal{T}_h}$ with $\omega \supset \omega_0$, while $\lambda_{\omega_0}(d_{\omega_0})$ is associated with $\omega_0 \in T_{\mathcal{T}_h}$. The computation of $\Lambda(u_h) = \lambda(d)$ starts with the recursive evaluation of Φ_ω for all $\omega \supset \omega_0$. Then the data d_{ω_0} are available and λ_{ω_0} can be applied.

4.4.7 Solution in a Subdomain

Suppose that the solution is only required in a small subdomain $\omega \in T_{\mathcal{T}_h}$. For this purpose the HDD method requires much less computational resources as usual. An example is shown in Fig. 4.13. The algorithm "Leaves to Root" is performed completely, but the algorithm "Root to Leaves" computes the solution only on the internal boundaries (dotted lines) which are necessary for computing the solution in ω. The storage requirements are also significantly reduced. We only store the mappings Φ^f_ω and Φ^g_ω for all $\omega \in T_{\mathcal{T}_h}$ that belong to the path from the root of $T_{\mathcal{T}_h}$ to ω. The storage requirement is $\mathcal{O}(n_h \log n_h)$, where n_h is the number of degrees of freedom in Ω. The computational cost of the "Root to Leaves" is $\mathcal{O}(n_h \log n_h)$ because the storage of an $\mathcal{H}$-matrix and as well as the $\mathcal{H}$-matrix - vector multiplication requires $\mathcal{O}(n_h \log n_h)$) (see Table 5.3).

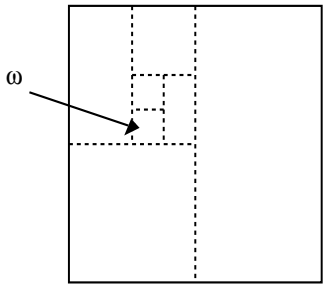


Figure 4.13: The solution in a subdomain $\omega \in T_{\mathcal{T}_h}$ is required. HDD computes the solution only on the dotted lines.

4.4.8 Homogeneous Problems

In the case of a zero right-hand side in (4.1), the mappings Ψ^f_ω and Φ^f_ω, $\omega \in T_{\mathcal{T}_h}$, do not need to be computed at all. Thus, only the mappings Ψ^g_ω and Φ^g_ω are of interest. The complexity of the HDD method in this case is $\mathcal{O}(k^2 n_h)$ and storage requirements $\mathcal{O}(k n_h)$, where n_h is the number of degrees of freedom in Ω. The application of the weak admissibility condition (see Subsection 5.5.2) results in multiplications of low-rank matrices and vectors for the algorithm "Root to Leaves". Numerical examples confirm the linear cost of the HDD method with a homogeneous right-hand side, instead of the "almost" linear cost $\mathcal{O}(n_h \log^q n_h)$ for an inhomogeneous right-hand side. In particular, for the discrete problem the CPU times for computing the solution on grids of sizes 257×257 and 513×513 with the relative error 10^{-3} are 37.0 sec. and 176.0 sec., correspondingly. The rate is $\frac{176.0}{37.0} = 4.76$ (factor 4 means a linear dependence).

5 Hierarchical Matrices

Almost all results of this Chapter were already published in [33], [30], [27], [11], [37], [29]. The new material is presented in Sections 5.9.5, 5.9.6, 5.9.7 and 5.10.

5.1 Introduction

The difficulty with the exact matrix arithmetic is that except for the diagonal matrices (or diagonal after a certain cheap transformation), there isn't a class of matrices which allows the standard matrix operations: Ax, $A+B$, $A \cdot B$, A^{-1} in $\mathcal{O}(N)$ operations.

The hierarchical matrices ($\mathcal{H}$-matrices) were introduced in 1999 by Hackbusch [33] and since then $\mathcal{H}$-matrices have been applied in a wide range of applications. They provide a format for the data-sparse representation of fully-populated matrices. The main idea in the $\mathcal{H}$-matrices is to approximate certain subblocks of a given matrix M by low-rank matrices. Let $R \in \mathbb{R}^{n\times m}$ be a subblock of M and rank(R)=k, $k \ll \min(n,m)$. Suppose that we find matrices $A \in \mathbb{R}^{n\times k}$ and $B \in \mathbb{R}^{m\times k}$ so that $R = AB^T$. The storage requirement for matrices A and B is $k(n+m)$ instead of $n\cdot m$ for matrix R. Later on will be shown that the cost of the basic matrix arithmetic (matrix-matrix addition, matrix-matrix multiplication, inversion of matrices) is not greater than $\mathcal{O}(n\log^{\alpha} n)$. One of the biggest advantages of $\mathcal{H}$-matrices is the almost linear complexity of the $\mathcal{H}$-matrix addition, multiplication and inversion.
In this section we give two examples of $\mathcal{H}$-matrices (see Fig. 5.1). The dark blocks are dense matrices and the light blocks are low-rank matrices. The steps in the grey blocks show the decay of the singular values in a logarithmic scale. The size of both matrices is 4096×4096. The first example is an $\mathcal{H}$-matrix approximation of the stiffness matrix of the Poisson problem for the grid as shown in Fig. 3.1. The second matrix is an $\mathcal{H}$-matrix approximation of the inverse of the stiffness matrix of the Poisson problem. The approximation error is $\|M_{\mathcal{H}}^{-1}M - I\|_2 = 2.1 \cdot 10^{-3}$.
In Section 5.4 we define auxiliary structures: clusters, cluster trees and the block cluster trees. After that we introduce the admissibility criterion and explain how to build a cluster tree and a block cluster tree. We give the definition of $\mathcal{H}$-matrices and define the low-rank arithmetic and the $\mathcal{H}$-matrix arithmetic. We describe how to convert one $\mathcal{H}$-matrix into another with a different block cluster tree. Finally, we estimate the computational complexities of the main arithmetic operations.

5.2 Notation

In this section we describe the most important notation which are used in the text. The finite index set $I := \{0, ..., n-1\}$ contains the indices of the basis functions b_i which are used in the Galerkin discretisation. We denote the cardinality of I by $n = |I|$.

5.3 $\mathcal{H}$-Matrix for an Elliptic Boundary Value Problem.

Let M be the stiffness matrix which comes from the problem (4.1). M is a data-sparse matrix, but M^{-1} is already a dense matrix. The next theorem proves the existence of the $\mathcal{H}$-matrix approximation of M^{-1}.

Theorem 5.3.1 *Let $\varepsilon_h > 0$ be the finite element error and $L = \mathcal{O}(\log n)$ the depth of the block cluster tree. Then there exists a hierarchical matrix $M_{\mathcal{H}}^{-1}$ with maximal rank $k = L^2 C_1 \log^{d+1}(\frac{LC_2}{\varepsilon_h})$ of the low-rank subblocks, such that*

$$\|M^{-1} - M_{\mathcal{H}}^{-1}\|_2 \leq C(1+\theta)\varepsilon_h,$$

where C_1, C are two constants, d the spatial dimension and $\theta \in (0,1)$.

Proof: see [13], [36].

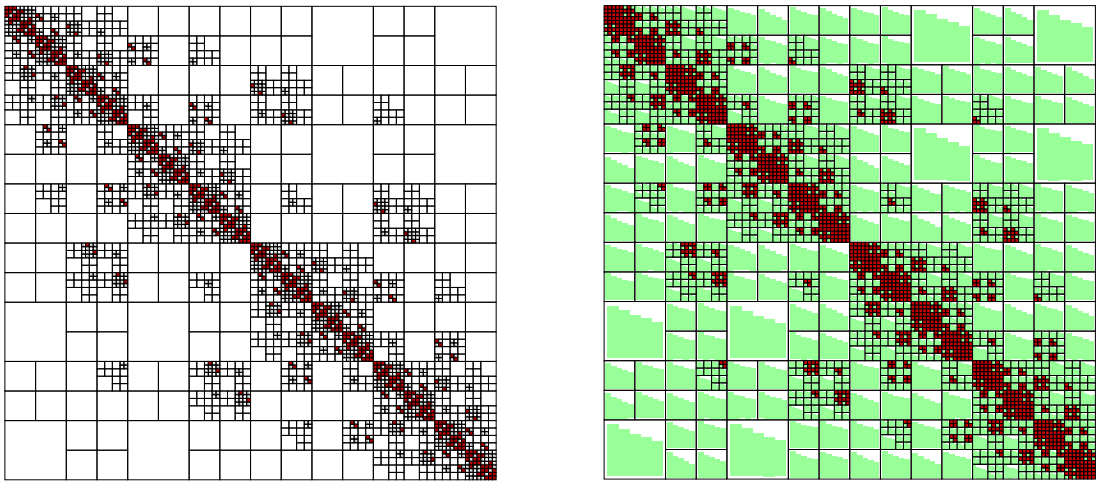


Figure 5.1: The $\mathcal{H}$-matrix approximation of the stiffness matrix of the Poisson problem (**left**) and its inverse (**right**). 64^2 dofs, $|M_{\mathcal{H}}^{-1}M - I|_2 = 2.1 \cdot 10^{-3}$. The dark blocks are dense matrices. The light blocks are low-rank matrices with maximal rank $k_{max} = 5$.

5.4 Building of $\mathcal{H}$-Matrices

In order to build an $\mathcal{H}$-matrix, we first need to introduce some auxiliary structures:

1. the cluster tree (denoted by T_I),
2. the block cluster tree (denoted by $T_{I\times I}$),
3. the admissibility condition (denoted by Adm_η or Adm_W).

5.4.1 Cluster Tree

We consider different partitions of I into disjoint subsets including coarse and fine partitions. The set of these partitions is hierarchically structured and is uniquely defined by the tree $T = T_I$, which is called *cluster tree.*

Notation 5.4.1 *We denote all vertices of a tree T by $V(T)$ and all sons of a vertex t by $S(t)$.*

Definition 5.4.1 *A vertex $v \in V(T)$ is a leaf if $sons(v) = \varnothing$ and we define*

$$\mathcal{L}(T) := \{v \in V(T) | \quad v \text{ is a leaf}\}.$$

Notation 5.4.2 *The uniquely determined predecessor (father) of a non-root vertex $v \in T_I$ is denoted by $\mathcal{F}(v)$.*

Definition 5.4.2 *We define the levels $l \in \mathbb{N}_0$ of T recursively*

$$T_I^{(0)} = \{I\}, \quad T^{(l)} := \{t \in T_I | \mathcal{F}(v) \in T_I^{l-1}\}$$

and we write level(v)=l if $v \in T_I^{(l)}$.
The leaves of T on level l are denoted by

$$\mathcal{L}(T, l) := T^{(l)} \cap \mathcal{L}(T).$$

Let the mapping $\hat{} : T_I \to \{r \mid r \subseteq I\}$ labels vertices of the tree T_I, i.e., if $t \in T_I$ and $\hat{t} \subseteq I$ then $\hat{} : t \mapsto \hat{t}$.
Thus, for each $t \in T_I$, we denote its label by $\hat{t} \subseteq I$.

Definition 5.4.3 *A finite tree T_I is a cluster tree over the index set I if the following conditions hold:*

- *I is the root of T and $\hat{t} \subseteq I$ holds for all $t \in T_I$.*
- *If $t \in T_I$ is not a leaf, then $S(t)$ contains disjoint subsets of I and $\hat{t}$ is the disjoint union of its sons, $\hat{t} = \bigcup_{s\in S(t)} \hat{s}$.*
- *If $t \in T_I$ is a leaf, then $|\hat{t}| \leq n_{min}$ for a fixed number n_{min}.*

Definition 5.4.4 *If $|S(t)| = 2$ for all $t \in T_I \setminus \mathcal{L}(T_I)$, then T_I is called a binary tree.*

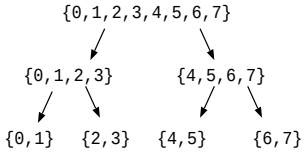


Figure 5.2: An example of a cluster tree over the index set $I = \{0, 1, 2, 3, 4, 5, 6, 7\}$. Each leaf contains $n_{min} = 2$ elements.

Definition 5.4.5 *The vertices $t \in V(T)$ of a cluster tree are called clusters.*

A cluster tree for I is usually denoted by T_I. We write $t \in T_I$ for $t \in V(T_I)$.
In [27], [13], [36] the reader can find two different algorithms for building cluster trees for a given set of basis functions. The first one is based on a geometrical splitting and the second one on a cardinality splitting. Which one to choose depends, for example, on the discretisation of a given problem.

Each index $i \in I$ is associated with the basis function b_i of the Galerkin ansatz space $V_h := span\{b_i\}_{i \in I}$, so that the support of the basis functions is denoted by

$$\Omega_i := \text{supp}(b_i) \quad \text{for} \quad i \in I.$$

We generalize Ω_t to cluster $t \in T_I$ by setting

$$\Omega_t := \bigcup_{i \in \hat{t}} \Omega_i, t \in T_I. \tag{5.1}$$

Construction of a cluster tree

Since dealing directly with the supports will be too complicated, we choose a point $x_i \in \Omega_i$ for each index $i \in I$ and work with these points instead of the supports. This simplification will not significantly harm the performance of the algorithm, since the supports of the typical finite element basis function is small.
Our construction of the cluster tree starts with the dense index set I, which is the root of the cluster tree by definition. After that we apply a suitable technique to find a disjoint partition of the index set and use this partition to create the son clusters. We apply the procedure recursively to the sons until the index sets are small enough. To make the partition easier we define for the given domain Ω_t from (5.1) a minimal *axes-parallel bounding box* $Q_t = \prod_{i=1}^{d} [a_i, b_i]$, where $Q_t \supset \Omega_t$. The coordinates a_i and b_i can be defined as following

$$\begin{aligned} &\text{for i=1 to d} \\ &\{ \\ &\quad a_i := \min_{j \in \hat{t}} x_{j,i} \\ &\quad b_i := \max_{j \in \hat{t}} x_{j,i} \\ &\} \end{aligned} \tag{5.2}$$

Each index $i \in \hat{t}$ corresponds to a point $x_i \in \mathbb{R}^d$. There are some variants of splitting the boundary box Q_t. For example, we can choose the coordinate direction of the maximal extent and split the box perpendicular to this direction into two subdomains. This gives us the partition $\{\hat{t}_0, \hat{t}_1\}$ of $\hat{t}$ (respectively $Q_t = Q_{t_1} \cup Q_{t_2}$).

Remark 5.4.1 *See the algorithms of splitting based on the geometry of Ω_t and the cardinality in Sections 5.5.2 , 5.5.3 in [36] and [13].*

The building of the special cluster tree T_I

The input data for the building of the cluster tree T_I is the index set I. In this work we use the geometrically balanced clustering (see other variants of the clustering in [28], [27]). This means that for building cluster trees we use the geometry of the given domain.
Let I, I_{11}, I_{12} and I_2 be given index sets such that $I = I_{11} \cup I_{12} \cup I_2$. Let $T_{I_{11}}$, $T_{I_{12}}$, T_{I_2} be given cluster trees, based on I_{11}, I_{12} and I_2. We want to build the binary cluster tree T_I such that I_{11}, I_{12}, $I_2 \in T_I$ (see Fig. 5.3). For this purpose we build cluster I_1 which has two sons I_{11}, I_{12} and cluster I which has sons I_1 and I_2.

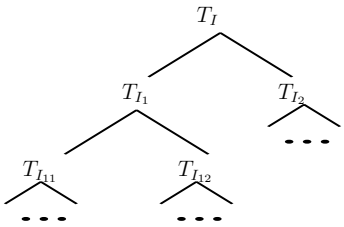


Figure 5.3: The structure of the cluster tree T_I.

5.4.2 Block Cluster Tree

While the vector components are indexed by $i \in I$, the entries of a matrix have indices from the index set $I \times J$. The block-cluster tree is nothing but a special cluster tree over the product index set $I \times J$. The vertices ("blocks") $b \in T_{I\times J}$ are of the form $b = (t, s)$ with $t \in T_I, s \in T_J$.

Definition 5.4.6 *Let T_I and T_J be cluster trees over the index sets I and J. A finite tree T is a block cluster tree based on T_I and T_J if the following conditions hold:*

- *$root(T) = I \times J$.*
- *Each vertex $b \in V(T)$ has the form $b = (t, s)$ for cluster $t \in T_I$ and $s \in T_J$.*
- *For each vertex $(t, s) \in V(T)$ with $sons(t, s) \neq \varnothing$, we have*

$$sons(t,s) = \begin{cases} (t,s') : s' \in sons(s), & \text{if } sons(t) = \varnothing \wedge sons(s) \neq \varnothing \\ (t',s) : t' \in sons(t), & \text{if } sons(t) \neq \varnothing \wedge sons(s) = \varnothing \\ (t',s') : t' \in sons(t), s' \in sons(s), & \text{otherwise} \end{cases} \tag{5.3}$$

- *The label of a vertex $(t, s) \in V(T)$ is given by $\widehat{(t,s)} = \widehat{t} \times \widehat{s} \subseteq I \times J$.*

A block cluster tree based on T_I and T_J is denoted by $T_{I\times J}$. We will use the abbreviation $(t,s) \in T_{I\times J}$ for $(t,s) \in V(T_{I\times J})$. We can see that $\widehat{root(T_{I\times J})} = I \times J$. This implies that the set of leaves $\mathcal{L}(T_{I\times J})$ is a partition of $I \times J$. An implementation of the block cluster tree can be found in [28].

Definition 5.4.7 *Let $T_{I\times J}$ be a block cluster tree. We call P a partition (or a block partition) of $I \times J$ if:*

$$\begin{gathered} P \subset T_{I\times J}, \\ b, b' \in P \Rightarrow (b = b' \text{ or } b \cap b' = \varnothing), \\ \bigcup_{b\in P} = I \times J. \end{gathered} \tag{5.4}$$

5.5 Admissibility

The admissibility condition helps us to find a balance between the storage requirements of an $\mathcal{H}$-matrix and its approximation accuracy. It also helps us to identify the blocks which can be approximated well by a low-rank matrices. Let $t \in T_I$, $s \in T_J$ and $t \times s \in T_{I\times J}$.

Definition 5.5.1 *The admissibility condition is a Boolean function*

$$Adm : T_{I\times J} \mapsto \{true, false\} \tag{5.5}$$

with the consistency requirement

$$Adm(b) \Rightarrow Adm(b') \quad \text{for all sons } b' \text{ of } b \in T_{I\times J}$$

and the property Adm(b)=true for all leaves $b \in T_{I\times J}$.

5.5.1 Standard Admissibility Condition (Adm_η)

Definition 5.5.2 *The standard admissibility criterion for $b = (t,s)$ is*

$$min\{diam(\Omega_t), diam(\Omega_s)\} \leq \eta dist(\Omega_t, \Omega_s), \tag{5.6}$$

where $\eta > 0$ is a fixed parameter.
The admissibility condition (5.5) now takes the form of

$$\begin{gathered} Adm_\eta(b) = true \quad \text{for} \quad b = (t,s) \in T_{I\times J} :\Longleftrightarrow \\ \text{(b is a leaf) or (5.6) holds.} \end{gathered}$$

In practice it is difficult to define the Euclidean diameter $diam(\Omega_t)$ and the Euclidean distance of two clusters $dist(\Omega_t, \Omega_s)$. This is the reason why we rewrite the standard admissibility criterion for the bounding boxes.
We define a minimal axis-parallel box $Q_t \subseteq \mathbb{R}^2$ such that $\Omega_t \subseteq Q_t$ holds.
This box will be called the *bounding box* of the cluster t.

Definition 5.5.3 *The standard admissibility criterion for Q_t and Q_s is:*

$$min\{diam(Q_t), diam(Q_s)\} \leq \eta \ dist(Q_t, Q_s).$$

Let d be the spatial dimension. We can compute the distances and diameters as follows:
If $Q_t = [a_1, b_1] \times ... \times [a_d, b_d]$ and $Q_s = [c_1, d_1] \times ... \times [c_d, d_d]$ then
$diam(Q_t) = \sqrt{\sum_{l=1}^{d} (b_l - a_l)^2}$, $diam(Q_s) = \sqrt{\sum_{l=1}^{d} (d_l - c_l)^2}$ and
$dist(Q_s, Q_t) = \sqrt{\sum_{l=1}^{d} dist([a_l, b_l], [c_l, d_l])^2}$.

Where does the admissibility condition come from?
Suppose the following propositions are true:

- $B_1 \subset \mathbb{R}^d$ and $B_2 \subset \mathbb{R}^d$ are compact.
- $\chi(x, y)$ is defined for $(x, y) \in B_1 \times B_2$ with $x \neq y$.

Let $\mathcal{K}$ be an integral operator with an asymptotic smooth kernel $\mathcal{K}$ in the domain $B_1 \times B_2$:

$$(\mathcal{K}v)(x) = \int_{B_2} \chi(x, y) v(y) dy \quad (x \in B_1).$$

Suppose that $\chi^{(k)}(x, y)$ is an approximation of χ in $B_1 \times B_2$ of the separate form:

$$\chi^{(k)}(x, y) = \sum_{\nu=1}^{k} \varphi_\nu^{(k)}(x) \psi_\nu^{(k)}(y),$$

where k is the rank of separation (index $^{(k)}$ is not a derivative!).
Then, under some conditions (see Paragraph 4.6.3 [36],[13]) we have

$$\|\chi - \chi^{(k)}\|_{\infty, B_1 \times B_2} \leq c_1 \left[\frac{c_2 \min\{diam(B_1), diam(B_2)\}}{dist(B_1, B_2)} \right]^k. \tag{5.7}$$

Proof: See Paragraph 4.6.3 in [36].
Now, if

$$\min\{diam(B_1), diam(B_2)\} \leq \frac{1}{c_2} dist(B_1, B_2). \tag{5.8}$$

there is an exponential convergence in (5.7).

Definition 5.5.4 *Let $\eta > 0$ and t, s be two clusters, Ω_t and Ω_s are supports of t and s. The block $b = (t, s)$ is called η-admissible, if*

$$\min\{diam(t), diam(s)\} \leq dist(t, s). \tag{5.9}$$

5.5.2 Weak Admissibility Condition (Adm_W)

Definition 5.5.5 *In the 1D case $Adm_W(b) = true$ for $b = (t, s) \in T_{I \times J} :\Leftrightarrow$ ((b is a leaf) or t and s are different clusters.*

In the following remark we list properties of the weak admissibility condition.

Remark 5.5.1

1. *The partitioning $\mathcal{P}_W$, obtained with the weak admissibility criterion Adm_W is simpler than the partitioning $\mathcal{P}_\eta$ obtained with the standard admissibility criterion Adm_η, i.e., subblocks are coarser and the number of subblocks is less (see example in Fig. (5.4)).*

2. *The weak admissibility yields a cheaper $\mathcal{H}$-matrix arithmetic (e.g. $\mathcal{H}$-matrix multiplication, $\mathcal{H}$-matrix inverse) compare with the standard admissibility.*

3. *$Adm_\eta \Longrightarrow Adm_W$, i.e., if a block $b = (t, s) \in T_{I\times J}$ is admissible w.r.t. the standard admissibility criterion, then block b is also admissible w.r.t. the weak admissibility criterion.*

4. *If $b = (t, s) \in T_{I\times J}$ is a weakly admissible block then the domains $\Omega_t = \bigcup_{i=1}^{|\hat{t}|} \operatorname{supp} b_i$ and $\Omega_s = \bigcup_{i=1}^{|\hat{s}|} \operatorname{supp} b_i$ can touch each other at most at a point.*

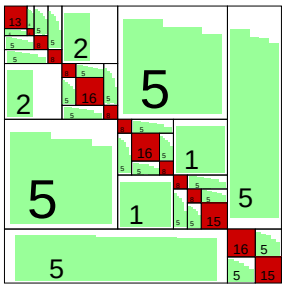


Figure 5.4: An example of a hierarchical matrix with weak admissible blocks.

Example 5.5.1 *Figure 6.1 shows an example of an $\mathcal{H}$-matrix which is obtained with the weak admissibility condition.*

Example 5.5.2 *Fig. 5.5 demonstrates three examples of clusters t and s. The block (t, s) in the cases (a) and (b) is weak admissible and in the case (c) inadmissible.*

See more about the weak admissibility condition in [37].

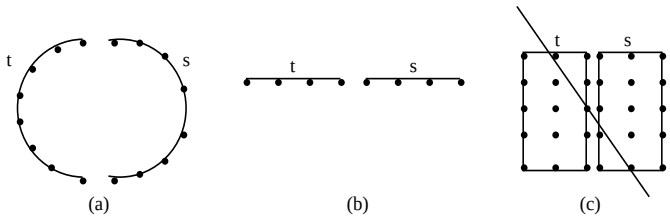


Figure 5.5: In (a),(b) the block (t, s) is weakly admissible and in (c) it is inadmissible.

Admissible Blocks

Let n_{min} be a given constant (in our numerical experiments $n_{min} = 32$).

Definition 5.5.6 *A block cluster tree $T_{I\times J}$ which is based on I and J, is called admissible with respect to an admissibility criterion if the following two conditions hold*

- (t, s) *is admissible,*
- $\min\{|\hat{t}|, |\hat{s}|)\} \leq n_{min}$

for all $(t, s) \in \mathcal{L}(T_{I\times J})$.

Notation 5.5.1 *We denote all admissible blocks by* $\mathcal{L}^+(T_{I\times J})$.

The construction of an admissible block cluster tree from the index sets I and J and a given admissibility condition is done in a straightforward recursion.

Algorithm 5.5.1 *(Building a block cluster tree)*

```
build_block_cluster_tree(t, s)
begin
  if (check_admissibility(t,s)=true) then
    if (min{|t̂|, |ŝ|} > n_min) then
      b :=create admissible node (t, s);
    else
      b :=create admissible leaf (t, s);
  else
    for each block (t', s') ∈ sons(t, s) do
      build_block_cluster_tree(t', s');
  end if;
end;
```

Example 5.5.3 *Figure 5.6 shows two index sets* $\hat{t} = I(\partial\omega)$ *and* $\hat{s} = I(\gamma_\omega)$. *These index sets do not have any common points. Let* $\Omega_t := \cup_{i\in\hat{t}} \operatorname{supp} b_i$ *and* $\Omega_s = \cup_{i\in\hat{s}} \operatorname{supp} b_i$. *The intersection of the supports* Ω_t *and* Ω_s *is not empty (see dark regions), but nevertheless the block* (t, s) *is weakly admissible (see [37]).*

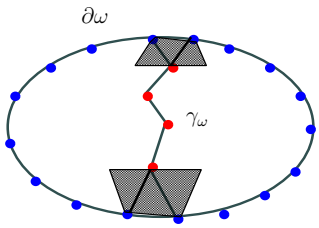


Figure 5.6: The intersection of supports Ω_s and Ω_t of two clusters t and s, where $\hat{t} = I(\partial\omega)$ and $\hat{s} = I(\gamma_\omega)$ is not empty (see dark regions), but nevertheless the weak admissible criterion can be applied and the block $b = (t, s)$ is weakly admissible.

Inadmissible Blocks

Definition 5.5.7 *A block (t, s) for which an admissibility criterion (standard, weak or some other) is false, is called inadmissible. We denote the set of all inadmissible blocks in $\mathcal{L}(T_{I\times J})$ by $\mathcal{L}^-(T_{I\times J})$.*

Recall that parameter n_{min} is responsible for the maximal size of inadmissible blocks. If a block $b = (t, s)$ does not satisfy to the first condition in Def. 5.5.6, but $\min\{|\hat{t}|, |\hat{s}|)\} \leq n_{min}$, then b is approximated by a dense matrix. If the parameter n_{min} is small (e.g. $n_{min} < 16$), then the matrix has a deeper hierarchy and the complexity of all arithmetic operations increases. On the other hand, if n_{min} is too large, the fully-populated matrix arithmetic begins to dominate.

5.6 Low-rank Matrix Format

Sometimes the name **rank-k matrix format** is used.
The rank-k matrices are very important for the construction of $\mathcal{H}$-matrices. We give the definition of rank-k matrices and we describe the low-rank arithmetic. Then we estimate the complexity of arithmetic operations and their implementation.

Let $M \in \mathbb{R}^{I\times J}$ be a matrix, I, J two index sets. Consider the factorization

$$M = AB^T \quad \text{where } A \in \mathbb{R}^{I\times\{1,\dots,k\}}, B \in \mathbb{R}^{J\times\{1,\dots,k\}}, k \in \mathbb{N}_0. \tag{5.10}$$

Definition 5.6.1 *We say that matrix M is **a rank-k matrix** if the representation (5.10) is given. We denote the class of all rank-k matrices for which factors A and B^T in (5.10) exist by $\mathcal{R}(k, I, J)$ or $\mathcal{R}(k, n, m)$, where $n := |I|$, $m := |J|$. If $M \in \mathcal{R}(k, I, J)$ we say that M has a **low-rank representation**.*

Remark 5.6.1 *If A,B and C are matrices and $A := B \cdot C$ then $Rank(A) \leq \min\{Rank(B), Rank(C)\}$. As a sequence we have $Rank(M) = Rank(AB^T) \leq k$ for matrices from (5.10).*

Remark 5.6.2 *Note that we do not state that the matrix M from the representation (5.10) has a rank k, but if $Rank(M) = k_0$ then there exists a factorization (5.10) with $k := k_0$.*

Proof: see Remark 2.2.2 in [36].

Definition 5.6.2 *$M \in \mathcal{R}(k, I, J) :\Longleftrightarrow M \in \mathbb{R}^{I\times J}$ is represented in format (5.10).*

Remark 5.6.3 *To store a matrix $M \in \mathcal{R}(k, n, m)$ we need $k(n+m)$ units of memory. A dense matrix $M \in \mathbb{R}^{n\times m}$ requires $n \cdot m$ units of memory. For $k \ll \min\{n, m\}$ the profit $nm - k(n + m)$ is especially remarkable.*

To introduce an approximate arithmetic in $\mathcal{R}(k, I, J)$ we recall the singular value decomposition.

Definition 5.6.3 *A dense matrix $M \in \mathbb{R}^{n\times m}$ has a singular value decomposition (SVD) $M = U\Sigma V^T$, if $U \in \mathbb{R}^{n\times n}$, $V \in \mathbb{R}^{m\times m}$ are unitary and $\Sigma \in \mathbb{R}^{n\times m}$ is diagonal with singular values Σ_{ii} (w.l.o.g. we can suggest that $\Sigma_{11} \geq \Sigma_{22} \geq ... \geq \Sigma_{nn}$, $n = min\{n, m\}$).*

To get the reduced singular value decomposition we omit all singular values, which are smaller than some level ε or we leave a fixed number of singular values (see Figure 5.7). After truncation we speak about *reduced singular value decomposition (denoted by rSVD)* $\tilde{M} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, where $\tilde{U} \in \mathbb{R}^{n\times k}$ contains the first k columns of U, $\tilde{V} \in \mathbb{R}^{m\times k}$ contains the first k columns of V and $\tilde{\Sigma} \in \mathbb{R}^{k\times k}$ contains the k-biggest singular values of Σ (see Fig. 5.7).

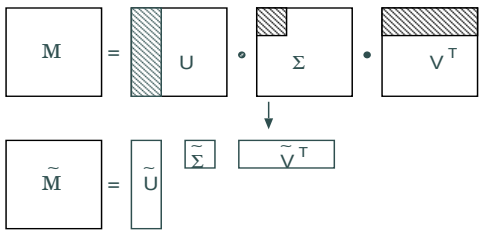


Figure 5.7: Reduced SVD, only k biggest singular values are taken.

Lemma 5.6.1 *Let $M \in \mathbb{R}^{n\times m}$ and $M = U\Sigma V^T$ (U, V orthogonal, Σ diagonal with singular values $\sigma_i = \Sigma_{ii}$ and $\sigma_1 \geq \sigma_2 \geq ...$). Then*

$$R := \tilde{U}\tilde{\Sigma}\tilde{V}^T \quad with \quad \tilde{\Sigma}_{ij} = \begin{cases} \sigma_i & for \quad i = j \leq \min\{k, n, m\}, \\ 0 & otherwise, \end{cases} \tag{5.11}$$

$\tilde{U} := U|_{n\times k}$, $\tilde{V} := V|_{m\times k}$ is the solution of the following two problems:

$$\min_{Rank(R)\leq k} \|M - R\|_2 \quad and \quad \min_{Rank(R)\leq k} \|M - R\|_F. \tag{5.12}$$

The errors are

$$\|M - R\|_2 = \sigma_{k+1} \quad and \quad \|M - R\|_F = \sqrt{\sum_{i=k+1}^{\min\{n,m\}} \sigma_i^2}. \tag{5.13}$$

Proof: see [51] or [25].

Let $M = AB^T \in \mathbb{R}^{n\times m}$ be a matrix in the rank-k matrix format, i.e. A and B are given. An rSVD $M = U\Sigma V^T$ can be computed efficiently in three steps:

1. Compute (reduced) a QR-factorization of $A = Q_A R_A$ and $B = Q_B R_B$, where $Q_A \in \mathbb{R}^{n\times k}$, $Q_B \in \mathbb{R}^{m\times k}$, and upper triangular matrices $R_A, R_B \in \mathbb{R}^{k\times k}$.
2. Compute an rSVD of $R_A R_B^T = U'\Sigma V'^T$.
3. Compute $U := Q_A U'$, $V := Q_A V'^T$.

For the realization of these steps the linear algebra packages LAPACK and BLAS are used. The first and third steps need $\mathcal{O}((n+m)k^2)$ operations and the second step needs $\mathcal{O}(k^3)$. The total complexity of rSVD is $\mathcal{O}((n+m)k^2 + k^3)$. Hence, we can compute rSVD of M with a linear complexity.
In HLIB one can find an implementation of this algorithm.
Now we introduce the rank-k matrix arithmetic. Later on we will use these operations for the definition of the $\mathcal{H}$-matrix arithmetic.

Lemma 5.6.2 *The product of a rank-k matrix and a dense matrix is again a rank-k matrix. Let $R \in \mathcal{R}(k,n,m)$, $N \in \mathbb{R}^{n'\times n}$ and $M \in \mathbb{R}^{m\times m'}$. Then*

$$NR \in \mathcal{R}(k,n',m), \quad RM \in \mathcal{R}(k,n,m'). \tag{5.14}$$

Proof: If $R = AB^T$ then $NR = (NA)B^T$ and $RM = A(M^T B)^T$.

Lemma 5.6.3 *Let $R_1 \in \mathcal{R}(k_1,n,m)$ and $R_2 \in \mathcal{R}(k_2,n,m)$. Then $R_1 + R_2 \in \mathcal{R}(k_1 + k_2, n, m)$.*

Proof: If $R_1 = AB^T$ and $R_1 = CD^T$ then $R_1 + R_2 = [AC][BD]^T$, where matrix $[AC] \in \mathbb{R}^{n\times(k_1+k_2)}$ and matrix $[BD] \in \mathbb{R}^{m\times(k_1+k_2)}$. Adding two rank-$k$ matrices does not require arithmetic operations, but the result matrix has a larger rank. This is why we introduce the operation *truncation*.

Definition 5.6.4 *(Truncation $\mathcal{T}_k$)*
Let $k' < k$ and $M \in \mathcal{R}(k,n,m)$ be a rank-k matrix. We define the truncation operator

$$\mathcal{T}^{\mathcal{R}}_{k'\leftarrow k} : \mathcal{R}(k,n,m) \to \mathcal{R}(k',n,m) \text{ by } \tilde{M} = \mathcal{T}^{\mathcal{R}}_{k'\leftarrow k}(M),$$

where $\tilde{M}$ is a best approximation of M in the set $\mathcal{R}(k,n,m)$ (not necessarily unique).

Notation 5.6.1 *If in Def. 5.6.4 the rank of the source matrix is not important we will write $\mathcal{T}_{k'}$.*

Lemma 5.6.4 *(Multiplication of rank-k matrices).*
Let $M_1 \in \mathcal{R}(k_1,I,J)$ and $M_2 \in \mathcal{R}(k_2,J,K)$ be given:

$$M_1 = A_1 B_1^T, \quad M_2 = A_2 B_2^T.$$

The product $M := M_1 M_2$ can be represented as
1. $M \in \mathcal{R}(k_2,I,K)$ with $A = A_1 B_1^T A_2$ and $B = B_2^T$. The computation of A costs $2k_1k_2(|I| + |J|) - k_2(|I| + k_1)$;
2. $M \in \mathcal{R}(k_1,I,K)$ with $A = A_1$ and $B = B_2 A_2^T B_1$. The computation of A costs $2k_1k_2(|J| + |K|) - k_1(|K| + k_2)$.

Proof: see ,e.g., [36].

Definition 5.6.5 *The formatted addition $\oplus$ in the set $\mathcal{R}(k,n,m)$ is*

$$A \oplus B := \mathcal{T}_k(A+B), \qquad A, B \in \mathcal{R}(k,n,m).$$

Remark 5.6.4 *Note that $\oplus$ is commutative, but in general not distributive (i.e., $(A \oplus B) \oplus C$ and $A \oplus (B \oplus C)$ may differ).*

Theorem 5.6.1 *Let $k \in \mathbb{N}$ be a maximal rank, and I, J, K be index sets. Table 5.1 shows storage requirements and computational complexities for rank-k matrices.*

Operation	Description	Complexity
storage(M)	$M \in \mathcal{R}(k,I,J)$, $M = AB^T$	$k(\|I\| + \|J\|)$
Mx	$M \in \mathcal{R}(k,I,J), M = AB^T, x \in \mathbb{R}^J$	$2k(\|I\| + \|J\|) - \|I\| - k$
$M' + M''$	$M' \in \mathcal{R}(k',I,J), M'' \in \mathcal{R}(k'',I,J)$	$(\|I\| + \|J\|)(k' + k'')$
$M'M''$	$M' \in \mathcal{R}(k',I,J), M'' \in \mathcal{R}(k'',I,J)$	$2k'k''(\|I\| + \|J\|) - k''(\|I\| + k')$
rSVD(M)	$M = AB^T \in \mathcal{R}(k,I,J)$	$6k^2(\|I\| + \|J\|) + 22k^3$
$\mathcal{T}^{\mathcal{R}}_{k' \leftarrow k}(M)$	$\mathcal{T}^{\mathcal{R}}_{k' \leftarrow k} : \mathcal{R}(k,I,J) \to \mathcal{R}(k',I,J)$	$6k^2(\|I\| + \|J\|) + 22k^3$
$M' \oplus_k M''$	$M', M'' \in \mathcal{R}(k,I,J)$	$24k^2(\|I\| + \|J\|) + 176k^3$

Table 5.1: Storage requirements and computational complexities for rank-k matrices.

Proof: see [33], [27], [13], [36].

5.7 Hierarchical Matrix Format

Definition 5.7.1 *Let $M \in \mathbb{R}^{I \times J}$ and $I' \times J'$ be a subset of $I \times J$, then the submatrix $M|_{I' \times J'} := (M_{i,j})_{(i,j) \in I' \times J'}$. For a superset $I'' \times J'' \supset I \times J$ we define the matrix $M'' := M|^{I'' \times J''} \in \mathbb{R}^{I'' \times J''}$ such that*

$$M''_{ij} := \begin{cases} M_{ij} & \text{if} \quad (i,j) \in I \times J, \\ 0 & \text{otherwise.} \end{cases} \tag{5.15}$$

Definition 5.7.2 *Let I and J be two finite index sets, $T := T_{I \times J}$ a block cluster tree, $\mathcal{P}$ a block partition, $k : \mathcal{P} \to \mathbb{N}$ a given mapping, n_{min} a small integer. The set of $\mathcal{H}$-matrices $\mathcal{H}(T,k) \subset \mathbb{R}^{I \times J}$ (with the partition $\mathcal{P}$ and the mapping k) consists of all $M \in \mathbb{R}^{I \times J}$ with*

$$rank(M|_b) \leq k(b) \quad \text{for all } b \in \mathcal{L}^+(T) \text{ and} \tag{5.16}$$

for all $b \in \mathcal{L}^+(T)$ the factors A, B in $M|_b = AB^T$ are given explicitly. The matrix blocks $b \in \mathcal{L}^-(T)$ are given in the standard full matrix representation and are small (the rank is smaller than n_{min}).

If $\forall b \in T\ k(b) = k$ then we speak about fixed rank, otherwise, about adaptive rank.

Let us write $A \in \mathcal{H}(T_{I\times J}, k, \mathcal{P})$ if we want to underline that matrix A has the partitioning $\mathcal{P}$ (we denote the weak partitioning by the subindex $_W$ and the standard partitioning by the subindex $_\eta$).

Definition 5.7.3 *Let $T_{I\times J}$ be a block cluster tree for the index sets I and J. A matrix $M \in \mathcal{H}(T_{I\times J}, k)$ is said to be stored in the $\mathcal{H}$-matrix representation if the submatrices corresponding to inadmissible leaves are stored as dense matrices and those corresponding to admissible leaves are stored in the rank-k matrix representation.*

To measure the sparsity property of a block cluster tree we introduce the following

Definition 5.7.4 *Let $T_{I\times J}$ be a block cluster tree based on T_I and T_J. We define the sparsity constant C_{sp} of $T_{I\times J}$ by*

$$C_{sp} := \max\{\max_{t\in T_I} |\{s|(t,s) \in \mathcal{L}(T)\}|, \max_{s\in T_J} |\{t|(t,s) \in \mathcal{L}(T)\}|\}.$$

Remark 5.7.1 *The complexities of all $\mathcal{H}$-matrix arithmetic operations depend on C_{sp}. For a model integral equation, which is discretised by BEM with the standard admissibility criterion, the constant C_{sp} takes values 3, 27, 189 for 1D, 2D, 3D problems accordingly.*

Remark 5.7.2 *Numerical experiments show that matrices from $\mathcal{H}(T_{I\times J}, k, \mathcal{P}_\eta)$ and $\mathcal{H}(T_{I\times J}, 3k, \mathcal{P}_W)$ give approximate similar accuracy. The storage and computational time for the matrices from $\mathcal{H}(\mathcal{P}_W, 3k)$ are smaller than the corresponding storage and time for $\mathcal{H}(\mathcal{P}_\eta, k)$.*

Lemma 5.7.1

1. *Let $A \in \mathcal{H}(T_{I\times J}, k_A, \mathcal{P}_W)$ and $B \in \mathcal{H}(T_{I\times J}, k_B, \mathcal{P}_W)$, then the exact product $A \cdot B$ belongs to $\mathcal{H}(T_{I\times J}, k_A + k_B, \mathcal{P}_W)$.*

2. *Let $A \in \mathcal{H}(T_{I\times J}, k, \mathcal{P}_W)$ be invertible, then $A^{-1} \in \mathcal{H}(T_{I\times J}, k, \mathcal{P}_W)$.*

3. *Let $A \in \mathcal{H}(T_{I\times I}, k, \mathcal{P}_W)$ and $I' \subset I$, then the Schur complement is $S_{I'} = A|_{I'\times I'} - A|_{I'\times I''} \cdot (A|_{I''\times I''})^{-1} \cdot A|_{I''\times I'}$, $I'' = I\backslash I'$, provided that $(A|_{I''\times I''})^{-1}$ exists.*

Proof: See [37].

Remark 5.7.3 *In HLIB we use the structures* **supermatrix** *for $\mathcal{H}$-matrices,* **rkmatrix** *for rank-k matrices and* **fullmatrix** *for dense matrices.*

5.8 Filling of Hierarchical Matrices

Suppose we have cluster trees T_I, T_J, $|I| > |J|$, a block cluster tree $T_{I\times J}$ and an admissibility criterion. Below we show how we compute elements of $M \in \mathcal{H}(T_{I\times J}, k)$ with complexity $\mathcal{O}(|I|\log|I|)$ (see more in [33], [29], [28]). Depending on the definition of M, different algorithms for the determinations of the matrix representations are needed.

Example 5.8.1 *1. The matrix M comes from an integral equation.*

2. The matrix M comes from a partial differential equation.

3. The matrix M is built from the $\mathcal{H}$-matrices M_1 and M_2 (our case).

5.8.1 $\mathcal{H}$-Matrix Approximation of BEM Matrix

Consider the following integral equation

$$\int_0^1 \log|x-y|U(y)dy = F(x), \qquad x \in (0,1).$$

After discretisation by Galerkin's method we obtain

$$\int_0^1\int_0^1 \phi_i(x)\log|x-y|U(y)dydx = \int_0^1 \phi_i(x)F(x)dx, \qquad 0 \le i < n,$$

in the space $V_n := span\{\phi_0, ..., \phi_{n-1}\}$, where ϕ_i, $i = 1, ..., n-1$, are some basis functions in BEM. The discrete solution U_n in the space V_n is $U_n := \sum_{j=0}^{n-1} u_j\phi_j$ with u_j being the solution of the linear system

$$Gu = f, \quad G_{ij} := \int_0^1\int_0^1 \phi_i(x)\log|x-y|\phi_j(y)dydx, \quad f_i := \int_0^1 \phi_i(x)F(x)dx. \tag{5.17}$$

We replace the kernel function $g(x,y) = \log|x-y|$ by a degenerate kernel

$$\tilde{g}(x,y) = \sum_{\nu=0}^{k-1} g_\nu(x)h_\nu(y). \tag{5.18}$$

Then we substitute $g(x,y) = \log|x-y|$ in (5.17) for $\tilde{g}(x,y)$

$$\tilde{G}_{ij} := \int_0^1\int_0^1 \phi_i(x)\sum_{\nu=0}^{k-1} g_\nu(x)h_\nu(y)\phi_j(y)dydx.$$

After easy transformations

$$\tilde{G}_{ij} := \sum_{\nu=0}^{k-1}\Big(\int_0^1 \phi_i(x)g_\nu(x)dx\Big)\Big(\int_0^1 h_\nu(y)\phi_j(y)dy\Big).$$

Now, all admissible blocks $G|_{(t,s)}$ can be represented in the form

$$G|_{(t,s)} = AB^T, \quad A \in \mathbb{R}^{|t|\times k}, \quad B \in \mathbb{R}^{|s|\times k},$$

where the entries of the factors A and B are

$$A_{i\nu} := \int_0^1 \phi_i(x) g_\nu(x) dx, \quad B_{j\nu} := \int_0^1 \phi_j(y) h_\nu(y) dy.$$

We use the fact that the basis functions are local and obtain for all inadmissible blocks:

$$\tilde{G}_{ij} := \int_{i/n}^{(i+1)/n} \int_{j/n}^{(j+1)/n} \log|x-y| dy dx.$$

5.8.2 $\mathcal{H}$-Matrix Approximation of FEM Matrix

The finite element discretisation of a partial differential equation leads to a data-sparse matrix with $\mathcal{O}(n)$ elements. For an elliptic equation

$$\begin{array}{ll} -\triangle u = g, & \text{in } \Omega \subset \mathbb{R}^d \\ u = 0, & \text{on } \partial\Omega \end{array}$$

the linear system is (see Section 3.4)

$$A\mathbf{u} = \mathbf{c}, \qquad A_{ij} := \int_\Omega \langle \nabla b_i(\mathbf{x}), \nabla b_j(\mathbf{x}) \rangle d\mathbf{x}, \qquad c_i := \int_\Omega b_i g d\mathbf{x},$$

where A is sparse. If the LU decomposition of A or A^{-1} is required, then A is converted to the $\mathcal{H}$-matrix format and the efficient H-matrix arithmetic is applied (see HLIB).

5.8.3 Building of an $\mathcal{H}$-Matrix from other $\mathcal{H}$-Matrices

Let $I', I'' \subset I$ and $J', J'' \subset J$ be index sets, $T'_{I'\times J'}$, $T''_{I''\times J''}$ and $T_{I\times J}$ be three block cluster trees. Let $M_1 \in \mathcal{H}(T'_{I'\times J'}, k_1)$, $M_2 \in \mathcal{H}(T''_{I''\times J''}, k_2)$ and $M \in \mathcal{H}(T_{I\times J}, k)$, then we can define

$$M := M_1|^{I\times J} + M_2|^{I\times J}.$$

For more details see Subsection 5.9.8.

5.9 Arithmetics of Hierarchical Matrices

In this section we describe the algorithms that perform the addition, multiplication and inversion in the hierarchical matrix format. The reader can get more information in [33], [29], [13] (English) and [27], [36] (German). Here, we also explain how to sum and multiply hierarchical matrices which have different block structures.

Remark 5.9.1 *In this Section and further, for simplicity of the notation, we pose that t and $\widehat{t}$ are equivalent and we do not separate between t and $\widehat{t}$. For instance, $|t| = |\widehat{t}|$, $M|_{t\times s} = M|_{(t,s)}$.*

In Section 5.6 we have defined the operator $\mathcal{T}^{\mathcal{R}}_{k'\leftarrow k}$, which truncates a rank-k matrix to the rank-k' matrix, $k' < k$. An extension of this operator to $\mathcal{H}$-matrices is as follows:

Definition 5.9.1 *(truncation of $\mathcal{H}$-matrices)*
Let $T := T_{I\times J}$ be a block cluster tree and $n := |I|$, $m := |J|$. Let $M \in \mathbb{R}^{n\times m}$, $M' \in \mathcal{H}(T,k)$. We define the truncation operator

$$\mathcal{T}^{\mathcal{H}}_k : \mathbb{R}^{n\times m} \to \mathcal{H}(T,k), \quad M \mapsto M',$$

where $M' = \mathcal{T}^{\mathcal{H}}_k(M)$ and $M'|_{(r,s)} = \mathcal{T}^{\mathcal{R}}_k(M|_{(r,s)})$ for all $(r,s) \in \mathcal{L}^+(T)$ and $M'|_{(r,s)} = M|_{(r,s)}$ for all $(r,s) \in \mathcal{L}^-(T)$.

Definition 5.9.2 *An alternative truncation operator $\mathcal{T}_\varepsilon$ is defined in the following way:*

$$\mathcal{T}_\varepsilon(M) := \operatorname{argmin}\{rank(R) | \frac{\|R-M\|_2}{\|M\|_2} \le \varepsilon\},$$

where the parameter ε is a desired accuracy.

Definition 5.9.3 *If the rank k of $M \in \mathcal{H}(T_{I\times J},k)$ (see Definition 5.7.2) is fixed a priory then we speak about **fixed rank arithmetic**. If the rank k depends on a block $b \in T_{I\times J}$ and is chosen as follows*

$$k = \min\{i : \sigma_i \le \varepsilon_a \sigma_1\},$$

*where $\{\sigma_i\}$ are the singular values of $M|_b$ then we speak about **adaptive rank arithmetic** (see [13] or Section 6 in [27]).*

The use of both truncation operators makes the matrix arithmetic more flexible.

Definition 5.9.4 *(Formatted addition of two hierarchical matrices)*
Let $A, B, C \in \mathcal{H}(T_{I\times J},k)$, $k \in \mathbb{N}$. The formatted addition of the matrices A and B is defined by

$$C := A \oplus B := \mathcal{T}_k(A+B).$$

If the rank k under consideration is not evident then we write $\oplus_k$ instead of $\oplus$.

Definition 5.9.5 *(Formatted multiplication of two hierarchical matrices)*
Let $A \in \mathcal{H}(T_{I\times J},k)$, $B \in \mathcal{H}(T_{J\times K},k)$, $C \in \mathcal{H}(T_{I\times K},k)$, $k \in \mathbb{N}$. The formatted multiplication of the matrices A, B is defined by

$$C = A \odot B := \mathcal{T}_k(A\cdot B).$$

The block structure of the $\mathcal{H}$-matrix product $A \cdot B$ does not retain either the structure of matrices A or B and can become rather complicated. See more details about $\mathcal{H}$-matrix multiplication in [13], [36] and [27].
Let J, J_1 and J_2 be three index sets and $J = J_1 \cup J_2$, $J_1 \cap J_2 = \varnothing$. Let $M \in \mathbb{R}^{I \times J}$, $M_1 \in \mathbb{R}^{I \times J_1}$ and $M_2 \in \mathbb{R}^{I \times J_2}$ be three matrices and $M = [M_1 M_2] \in \mathbb{R}^{I \times J}$. The formatted agglomeration of matrices M_1 and M_2 is defined by:

$$[M_1 M_2] := M_1|^{I \times J} + M_2|^{I \times J}. \tag{5.19}$$

Definition 5.9.6 *Let $k_1, k_2 \in \mathbb{N}_0$, $M_1 \in \mathcal{R}(k_1, I, J_1), M_2 \in \mathcal{R}(k_2, I, J_2)$ and $J = J_1 \cup J_2$, $J_1 \cap J_2 = \varnothing$. Then $M = \mathcal{T}^{\mathcal{R}}_{k \leftarrow k_1 + k_2}([M_1 M_2]) = M_1|^{I \times J} \oplus_k M_2|^{I \times J} \in \mathcal{R}(k, I, J)$ is called formatted agglomeration.*

In the general case there are more than two terms of the agglomeration.

Definition 5.9.7 *Let $M_i \in \mathcal{R}(k_i, I, J)$, $i = 1, ..., q$. The operation*

$$M := \mathcal{T}^{\mathcal{R}}_{k \leftarrow \sum_{i=1}^{q} k_i}(\sum_{i=1}^{q} M_i) \tag{5.20}$$

is the truncated agglomeration of q terms.

Lemma 5.9.1 *If in (5.20) $k_1 = k_2 = ... = k_q = k$, then the complexity of the truncated agglomeration of q terms is $\mathcal{O}(k^2 q^2 n)$, $n = max\{|I|, |J|\}$.*

Proof: The cost of the truncation $\mathcal{T}_{k \leftarrow 2k}$ is $\mathcal{O}((2k)^2 n)$. For q terms the cost is $\mathcal{O}((kq)^2)n) = \mathcal{O}(k^2 q^2 n)$.

Remark 5.9.2 *A second possibility of a truncated agglomeration for $q > 2$ is the pairwise truncation:*

$$M = \mathcal{T}^{\mathcal{R}}_{k \leftarrow k_1 + k} M_1 + ... + \mathcal{T}^{\mathcal{R}}_{k \leftarrow k_{q-2} + k}(M_{q-2} + \mathcal{T}^{\mathcal{R}}_{k \leftarrow k_{q-1} + k_q}(M_{q-1} + M_q))...). \tag{5.21}$$

Lemma 5.9.2 *The complexity of the truncated addition as in (5.21) is*

$$\mathcal{O}(k^2_{max}(q-1)n), \tag{5.22}$$

where $k'_{max} := \max\{k_i + k |\, i = 1, ..., q-2\}$ and $k_{max} := \max\{k'_{max}, k_{q-1} + k\}$.

Proof: The cost of the truncation $\mathcal{T}_{k_i \leftarrow k_{i+1} + k_{i+2}}$ is $\mathcal{O}((k_{i+1} + k_{i+2})^2 n)$.
Let $k_{max} := k_i + k_{i+1}$, $i = 1, ..., q-1$. For q terms the cost is $\mathcal{O}(k^2_{max}(q-1)n)$.

Remark 5.9.3 *The pairwise truncated addition of q terms is cheaper than the direct truncated addition (5.20) of q terms. For $k_1 = ... = k_q$ the profit is*

$$(q^2 \cdot k^2 - (2k)^2 \cdot (q-1))n = (q^2 - 4q + 4)k^2 n.$$

But practical experiments show that the accuracy of the pairwise truncated addition is worse than the truncated addition of q terms.

Remark 5.9.4 *It is also may be possible that the following truncated addition*

$$M = \mathcal{T}^{\mathcal{R}}_{k \leftarrow k_1 + k_2 + k}(M_1 + M_2 + ... + \mathcal{T}^{\mathcal{R}}_{k \leftarrow k_{q-4} + k_{q-3} + k}(M_{q-4} + M_{q-3} + \mathcal{T}^{\mathcal{R}}_{k \leftarrow k_{q-2} + k_{q-1} + k_q}(M_{q-2} + M_{q-1} + M_q))...).$$

is cheaper than the pairwise truncated addition.

5.9.1 Matrix - Vector Multiplication

Let $M \in \mathcal{H}(T_{I\times J}, k)$, $n = |I|$, $m = |J|$, $v \in \mathbb{R}^m$ and $w \in \mathbb{R}^n$. The matrix-vector (denote by MV) multiplication $w = Mv$ is realized in a recursive way. The procedure $MV(M[i], v[i], w[i])$, where $M[i]$, $v[i]$ and $w[i]$ are corresponding parts of M, v and w calls itself recursively.

Remark 5.9.5 *Implementation of MV multiplication in HLIB is* $eval\_supermatrix(M, v, w)$.

5.9.2 Matrix - Matrix Multiplication

The $\mathcal{H}$-matrix multiplication (denote by MM) $A \cdot B = C$, where $A \in \mathcal{H}(T_{I\times J}, k)$, $B \in \mathcal{H}(T_{J\times K}, k)$ and $C \in \mathcal{H}(T_{I\times K}, k)$ is realized block-wise recursively. Suppose that A and B are 2×2 blocks matrices, then

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \mathcal{T}_k \begin{pmatrix} A_{11} \odot B_{11} \oplus A_{12} \odot B_{21} & A_{11} \odot B_{12} \oplus A_{12} \odot B_{22} \\ A_{21} \odot B_{11} \oplus A_{22} \odot B_{21} & A_{22} \odot B_{22} \oplus A_{21} \odot B_{12} \end{pmatrix}.$$

Remark 5.9.6 *In the case when the matrices A and B have different block structures the $\mathcal{H}$-matrix multiplication of A and B is possible after conversion of A or B to the respective format.*

Thus, the product of two $\mathcal{H}$-matrices or their sum can require the truncation of the rank.

Lemma 5.9.3 *(complexity of the $\mathcal{H}$-matrix truncation)*
Let $T := T_{I\times J}$ be a block cluster tree, based on the cluster trees T_I and T_J. A truncation $\mathcal{T}^{\mathcal{H}}_{k'\leftarrow k}(M)$ of $M \in \mathcal{H}(T_{I\times J}, k)$ can be computed with the complexity

$$N_{k'\leftarrow k} \leq 6kN_{St}(T, k) + 23k^3|\mathcal{L}(T)|,$$

where N_{St} is the storage requirement for M.

Proof: see Lemma 2.9 in [29].

5.9.3 Hierarchical Approximation $\mathcal{T}^{\mathcal{R}\leftarrow\mathcal{H}}_k$

The hierarchical approximation is applied for the MM multiplication and for the MM conversion.

Notation 5.9.1 *We denote an operator which truncates a dense matrix $M \in \mathbb{R}^{I\times J}$ to a rank-k matrix by $\mathcal{T}^{\mathcal{R}\leftarrow\mathcal{F}}_k$.*

Note that $\mathcal{T}^{R\leftarrow\mathcal{F}}_k(M)$ is done by the singular value decomposition (see Section 5.6).

Notation 5.9.2 *We denote the operator, which hierarchically convert $M \in \mathcal{H}(T_{I\times I}, k)$ to a rank-k matrix in $p + 1$ steps (see Fig. 5.8) by $\mathcal{T}^{\mathcal{R}\leftarrow\mathcal{H}}_k$.*

Definition 5.9.8 *(Hierarchical Approximation) Let $T_{I\times I}$ be a block cluster tree, $p := depth(T_{I\times I})$, $M \in \mathcal{H}(T_{I\times I}, k)$. Let P be a partitioning. We define the hierarchical approximation $M_{\mathcal{H}}$ of M in $p+1$ steps as follows:*

$$\mathcal{T}_k^{\mathcal{R}\leftarrow\mathcal{H}} := \begin{cases} \mathcal{T}_k^{\mathcal{R}\leftarrow\mathcal{F}}(M|_b) & \text{if } b \in \mathcal{L}(T_{I\times J}) \wedge b \text{ is inadmissible,} \\ \mathcal{T}_k^{\mathcal{R}}(M|_b) & \text{if } b \in \mathcal{L}(T_{I\times J}) \wedge b \text{ is admissible,} \\ \mathcal{T}_{k\leftarrow k\cdot|S(b)|}^{\mathcal{R}}(M|_b) & \text{if } b \in P \wedge b \notin \mathcal{L}(T_{I\times J}), \end{cases}$$

where $|S(b)|$ is the number of sons of b.

The hierarchical approximation (see Fig. 5.8) contains two subprocedures:

1. The conversion a given dense matrix to a rank-k matrix ($addfull2\_rkmatrix(..)$ in HLIB).

2. Adding two or more rank-k matrices with truncation to a rank-k matrix ($addparts2\_rkmatrix(..)$ in HLIB).

Lemma 5.9.4 *Let $M \in \mathcal{H}(T_{I\times J}, k)$, $R \in \mathcal{R}(k, I, J)$. The complexity of the truncation $R = \mathcal{T}_k^{\mathcal{R}\leftarrow\mathcal{H}}(M)$ is $\mathcal{O}(k^2 n \log n)$, where $n = \max\{|I|, |J|\}$.*

Proof: see Lemma 6.4.4 in [36].

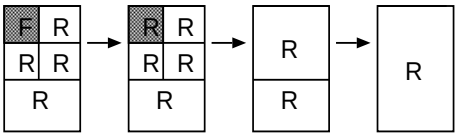


Figure 5.8: Three steps of the hierarchical approximation. The first step is the conversion of a dense matrix F to a rank-k matrix R. In the second step four rank-k matrices are converted to a larger rank-k matrix. In step three two low-rank matrices are converted to a global rank-k matrix.

Lemma 5.9.5 *(Hierarchical Approximation Error)*
Let $p := depth(T_{I\times J})$, $M \in \mathcal{H}(T_{I\times J}, k)$ and $M_{\mathcal{H}}$ be a hierarchical approximation of M, then

$$\|M - M_{\mathcal{H}}\|_F \leq (2^{p+1} + 1)\|M - \mathcal{T}_k^{\mathcal{H}}(M)\|_F. \tag{5.23}$$

$$\|M - M_{\mathcal{H}}\|_2 \leq (2^{\frac{3p}{2}+1} + 2^{\frac{p}{2}})\|M - \mathcal{T}_k^{\mathcal{H}}(M)\|_2.$$

Proof: see [27], [13], [36].

5.9.4 $\mathcal{H}$-Matrix Inversion

Theorem 5.9.1 *Let $M \in \mathcal{H}(T_{I\times I}, k)$ be an $\mathcal{H}$-matrix with parameter $n_{min} = k$. The block cluster tree $T_{I\times I}$ is based on a binary cluster tree T_I and for all $(r, s) \in T_{I\times I}$ we define*

$$S(r \times s) = \begin{cases} \{r' \times s' | r' \in S(r), s' \in S(s)\} & \text{if } r = s, \\ \varnothing & \text{otherwise.} \end{cases} \tag{5.24}$$

Let M be invertible and $p := depth(T_{I\times I})$. Then the exact inverse M^{-1} to M fulfils

$$M^{-1} \in \mathcal{H}(T_{I\times I}, kp).$$

Proof: *see Section 3.1 in [29].*

Recursive formula

Let $A \in \mathcal{H}(T_{I\times I}, k)$ be given as a 2×2 block matrix. Let A and A_{11} be regular matrices. Then the inversion of

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \tag{5.25}$$

can be computed by the following recursive formula:

$$A^{-1} = \begin{pmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}S^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}S^{-1} \\ -S^{-1}A_{21}A_{11}^{-1} & S^{-1} \end{pmatrix}, \tag{5.26}$$

where $S := A_{22} - A_{21}A_{11}^{-1}A_{12}$. This formula is obtained by block Gauss-elimination. Here all arithmetic operations are done in the $\mathcal{H}$-matrix format. See more [36], [13].

Remark 5.9.7 *The matrix inversion is cheaper if the partitioning is obtained with the weak admissibility criterion. The reason is that the Schur complement S^{-1} in (5.26) can be computed with the use of the Sherman-Morrison-Woodbury formula (see more in [37]).*

$\mathcal{H}$-matrix Inversion by the DD Method

One can use the domain decomposition idea for the inversion (see Section 4.1).

$\mathcal{H}$-matrix Inversion by LU decomposition

Numerical results show that inversion by a hierarchical LU decomposition (denote by $\mathcal{H}$-LU) is faster than the recursive formula (5.26). Here we briefly describe the $\mathcal{H}$-LU decomposition of a matrix A (see [48],[9],[47]). Assume that all minors of A are non-zero, then A can be decomposed in a product of a lower triangular matrix L and an upper triangular matrix U. L and U can be approximated by the $\mathcal{H}$-matrices $L_{\mathcal{H}}$ and $U_{\mathcal{H}}$ if any Schur complement in matrix A has this property (see proof in [10]).

$A \approx L_{\mathcal{H}} U_{\mathcal{H}}$ implies $A^{-1} \approx U_{\mathcal{H}}^{-1} L_{\mathcal{H}}^{-1}$.
Suppose that
$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}.$$
If A is a small dense matrix we use the standard pivoted LU decomposition. Otherwise we proceed as follows:

1. Compute L_{11} and U_{11} as $\mathcal{H}$-LU decomposition of A_{11}.

2. Compute U_{12} from $L_{11}U_{12} = A_{12}$ (use a recursive block forward substitution).

3. Compute L_{21} from $L_{21}U_{11} = A_{21}$ (use a recursive block backward substitution).

4. Compute L_{22} and U_{22} as $\mathcal{H}$-LU decomposition of $L_{22}U_{22} = A_{22} - L_{21}U_{12}$.

All steps are executed in the class of $\mathcal{H}$-matrices.
The complexity of the $\mathcal{H}$-LU decomposition is $\mathcal{O}(n \log^2 n)$. See for more theory and numerical experiments in [9]. One can find realizations of the $\mathcal{H}$-LU and the $\mathcal{H}$-Cholesky decompositions in HLIB.

Remark 5.9.8 *If the initial matrix A is symmetric (i.e., $L = U^H$) then we compute the $\mathcal{H}$-Cholesky decomposition.*

5.9.5 Other Operations With an $\mathcal{H}$-Matrix

Let $M \in \mathcal{H}(T_{I\times J}, k)$, $n := |I|$ and $m := |J|$. The following operations can be performed recursively.
1) Extracting a column (row) from M,
2) The removal of a column (row) from M,
3) Adding a rank-1 matrix to M.

Remark 5.9.9 *In order to delete a column i from a rank-k matrix $R = AB^T$, one should delete the column i from the matrix B^T.*

5.9.6 Extracting a Part of an $\mathcal{H}$-Matrix

Let I, J, I' and J' be four index sets, T_I and T_J be two cluster trees, $T := T_{I\times J}$, $M \in \mathcal{H}(T_{I\times J}, k)$. Let $t \in T_I$, $s \in T_J$ two clusters such that $I' \subseteq t$ and $J' \subseteq s$. The problem is to extract $M|_{I'\times J'}$.

$M|_{I'\times J'}$ can be a) a fully populated matrix, b) a rank-k matrix, and c) an $\mathcal{H}$-matrix (see an example in Fig. 5.10).
In Case (a) a part of a fully populated submatrix should be copied (is evident).
Case (b). Let $M|_{t\times s} = AB^T \in \mathcal{R}(k, t, s)$ be a rank-k matrix. The restriction $R' = R|_{I'\times J'}$ of the matrix R is also a rank-k matrix with the rank $k' = \min(k, |I'|, |J'|)$ (see Figure 5.9) and $R' := A|_{I'}B|_{J'}^T$.
Case (c). If $M|_{t\times s}$ is an $\mathcal{H}$-matrix, then the index sets I' and J' define the restriction of the block cluster tree $\tilde{T} := T|_{I'\times J'}$ (see Section 5.9.7).

Remark 5.9.10 *The following procedures are used in HLIB for extracting a part of an $\mathcal{H}$-matrix:*
1. *addpart_fullmatrix(..) copies a part of a dense matrix (see HLIB).*
2. *addpart2_rkmatrix(..) copies a part of a rank-k matrix (see HLIB).*
3. *cut_Hmatrix(..) copies a part of an $\mathcal{H}$-matrix (see HDD package).*

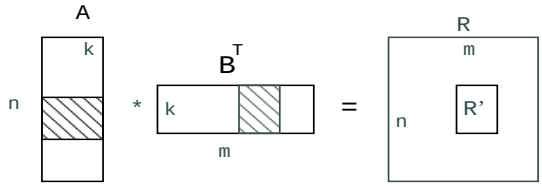


Figure 5.9: A part R' of a rank-k matrix $R = AB^T$ is the product of $A|_{I'}$ and $B^T|_{J'}$.

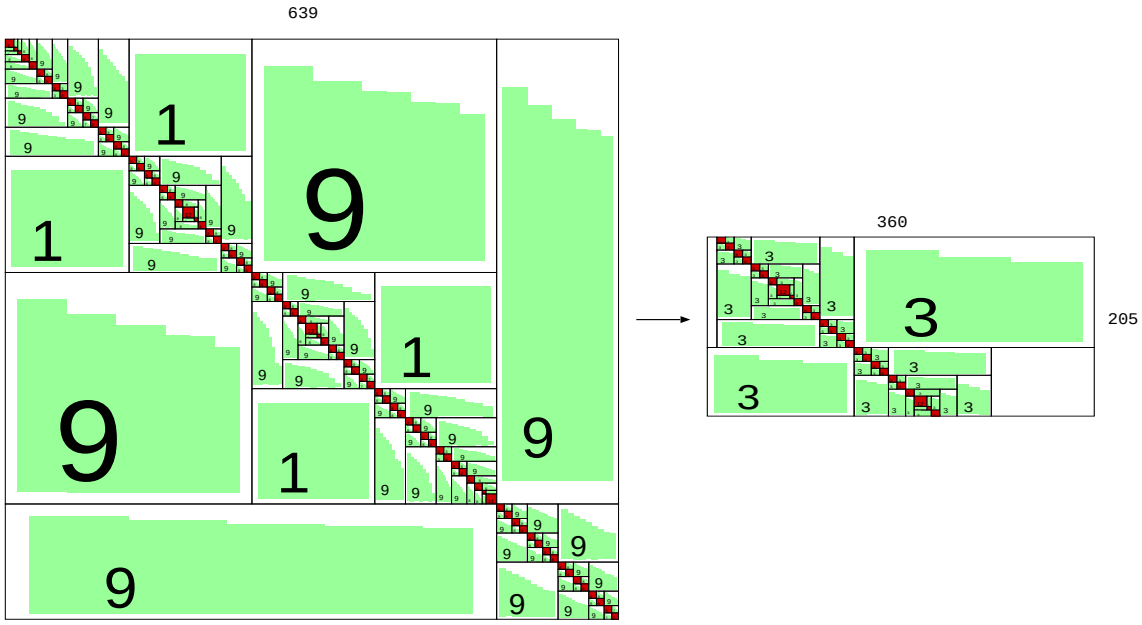


Figure 5.10: Starting from the position (130,120), we take 205 rows and 360 columns from $H \in \mathbb{R}^{639\times639}$. $H' \in \mathbb{R}^{205\times360}$ is a part of the matrix H.

Another way of extracting a submatrix $M' \in \mathbb{R}^{I'\times J'}$ from an $\mathcal{H}$-matrix $M \in \mathbb{R}^{I\times J}$, $I' \subseteq I$, $J' \subseteq J$, is demonstrated in Fig. 5.11. The idea is to multiply the original $\mathcal{H}$-matrix M from left and from right on the special matrices $T_1 \in \mathbb{R}^{I'\times J}$ and $T_2 \in \mathbb{R}^{I\times J'}$. The lack of this method is that the result matrix will be in the dense matrix format.

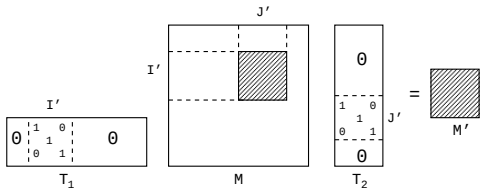


Figure 5.11: The submatrix $M' := M|_{I' \times J'} = T_1 \cdot M \cdot T_2$, where $I' \subseteq I$ and $J' \subseteq J$
.

5.9.7 Matrix - Matrix Conversion

Let $T := T_{I \times J}$, $I' \subseteq I$ and $J' \subseteq J$. First, we introduce the restricted block cluster tree $\tilde{T} := T|_{I' \times J'}$. We build the tree $\tilde{T}$ in two steps:

1. Create copy $\tilde{T}$ of T.
2. Each block $b := t \times s \in T$ becomes $\tilde{b} = (t \cap I', s \cap J')$. Note that the root of $\tilde{T}$ is $I' \times J'$. $\tilde{T}$ may contain nodes with the empty index set.

Algorithm 5.9.1 becomes the matrix $M \in \mathcal{H}(T_{I \times J}, k)$ and two index sets I' and J' and computes the matrix $\tilde{M} := M|_{I' \times J'}$.

Algorithm 5.9.1 *(Computing $M|_{I' \times J'}$, where $M \in \mathcal{H}(T_{I \times J}, k)$)*
***extract_part_Hmatrix**(M, I', J')*
begin
  if** (M is a dense matrix) **then
    allocate memory for a new dense matrix $F \in \mathbb{R}^{I' \times J'}$;
    $F := M|_{I' \times J'}$;
    ***return** F;*
  ***end if**;*
  if** (M is a rank-k matrix) **then
    allocate memory for a new rank-k matrix $R \in \mathcal{R}(k, I', J')$;
    $R := M|_{I' \times J'} = A|_{I'} B|_{J'}^T$;
    ***return** R;*
  ***end if**;*
  if** (M is an $\mathcal{H}$-matrix) **then
   for each** subblock $b = t \times s$ of M **do
     if** $(I' \cap t \neq \varnothing)$ and $(J' \cap s \neq \varnothing)$ **then
       $P :=$ extract_part_Hmatrix($M|_b$, $I' \cap t$, $J' \cap s$);
       ***return** P;*
     ***end if**;*
  ***end if**;*
***end**;*

Consider a more difficult case. Suppose $M \in \mathcal{H}(T_{I\times J}, k)$,

$$I \supseteq I' = \bigcup_{i=1}^{p} I_i, \quad I_j \cap I_k = \varnothing, j \neq k, \tag{5.27}$$

$$J \supseteq J' = \bigcup_{j=1}^{q} J_j, \quad J_i \cap J_k = \varnothing, i \neq k \tag{5.28}$$

and $n = \max\{|I|, |J|\}$, $n' = \max\{|I'|, |J'|\}$. Let $\tilde{M} \in \mathcal{H}(T'_{I'\times J'}, k)$. The problem is to convert M to $\tilde{M}$. We set up $\tilde{M}$ by Algorithm 5.9.2.

Algorithm 5.9.2 *(Conversion $M \in \mathcal{H}(T_{I\times J}, k)$ to $M' := M|_{I'\times J'} \in \mathcal{H}(T'_{I'\times J'}, k)$)*
h2h$(M,\ M',\ \bigcup_{i=1}^{p} I_i,\ \bigcup_{j=1}^{q} J_j)$
begin
  if *(M' is a dense matrix)* ***then***
    h2f($M,\ M',\ \bigcup_{i=1}^{p} I_i,\ \bigcup_{j=1}^{q} J_j$*);*
  if *(M' is a rank-k matrix)* ***then***
    h2r($M,\ M',\ \bigcup_{i=1}^{p} I_i,\ \bigcup_{j=1}^{q} J_j$*);/*see Algorithm 5.9.3*/*
  if *(M' is an $\mathcal{H}$-matrix)* ***then***
    for each *subblock $b = t \times s$ of M'* ***do***
      h2h($M,\ M'|_b,\ \bigcup_{i=1}^{p}(I_i \cap t),\ \bigcup_{j=1}^{q}(J_j \cap s)$*);*
  end if*;*
end*;*

The conversion of an $\mathcal{H}$-matrix to a dense matrix (procedure *h2f(..)* in Algorithm 5.9.2) is done elementwise. The conversion of an $\mathcal{H}$-matrix to a rank-k matrix (procedure *h2r(..)* in Algorithm 5.9.2) is done by Algorithm 5.9.3.

Algorithm 5.9.3 *(Converting $M \in \mathcal{H}(T_{I\times J}, k)$ to $R \in \mathcal{R}(k, I', J')$)*

h2r$(M,\ R,\ \bigcup_{i=1}^{p} I_i,\ \bigcup_{j=1}^{q} J_j)$
begin
  if *(M is a dense matrix)*
    allocate memory for $F \in \mathbb{R}^{I'\times J'}$;
    for all $i \in I'$ *and* $j \in J'$ ***do***
      $F_{ij} := M_{ij}$;
    convert F to R; / SVD is used */*
  end if;
  if *(M is a rank-k matrix, i.e. $M = AB^T$)* ***then***
    allocate memory for $R = CD^T \in \mathcal{R}(k, I', J')$;
    $C := A,\ D := B$;
  end if;
  if *(M is an $\mathcal{H}$-matrix)* ***then***
    $l = 0$;
    for each *subblock $b = t \times s$ of M* ***do***
      $R[l]$:=*h2r*$(M|_b,\ \bigcup_{i=1}^{p}(I_i \cap t),\ \bigcup_{j=1}^{q}(J_j \cap s))$;
      $l++$;
    end for
    R:=$(R[0] \oplus_k (R[1] \oplus_k ... \oplus_k (R[l-2] \oplus_k R[l-1])..)$;
    /*see pairwise truncation in (5.21)*/
  end if;
end;

5.9.8 Adding Two $\mathcal{H}$-Matrices With Different Block Cluster Trees

The addition of two hierarchical matrices with compatible block cluster trees has been described in [33]. Let I, J, I', J', I'' and J'' be given index sets such that $I', I'' \subseteq I$, $J', J'' \subseteq J$, and $M \in \mathcal{H}(T_{I\times J}, k)$. Let $T_{I'\times J'}$ and $T_{I''\times J''}$ be block cluster trees. The sum of $M_1 \in \mathcal{H}(T_{I'\times J'}, k_1)$ and $M_2 \in \mathcal{H}(T_{I''\times J''}, k_2)$ with the result matrix M is defined as follows (see Fig. 5.12):

$$M = M' \oplus M'', \quad \text{where } M' := M_1|^{I\times J} \text{ and } M'' := M_2|^{I\times J} \quad \text{(see Def. 5.7.1).}$$

The adding procedure applies the list of procedures from Table 5.2.

Remark 5.9.11 *Note that $M_1|^{I\times J}$ and $M_2|^{I\times J}$ have the block cluster tree $T_{I\times J}$. To compute $M' := M_1|^{I\times J}$ and $M'' := M_2|^{I\times J}$ we apply Algorithm 5.9.2.*

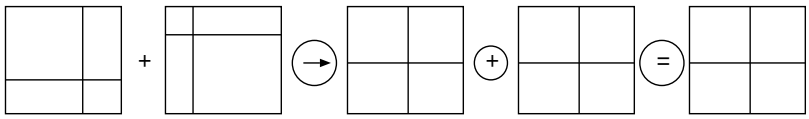


Figure 5.12: Transformation of $\mathcal{H}$-matrices M_1, M_2 to $\mathcal{H}$-matrices $M_1|^{I\times J}$, $M_2|^{I\times J}$ and their addition.

procedure	description
add_fullmatrix(F, F_1, F_2)	Adding two dense matrices
add_rkmatrix(R, R_1, R_2)	Adding two rank-k matrices
addfullpart2_rkmatrix(F,R)	Addition of a dense matrix to a rank-k matrix
addrk2_fullmatrix(R,F)	Addition of a rank-k matrix to a dense matrix
addfull2_supermatrix(F, M)	Addition of a dense matrix to an $\mathcal{H}$-matrix
addrk2_supermatrix(R, M)	Addition of a rank-k matrix to an $\mathcal{H}$-matrix
add_supermatrix(M, M_1, M_2)	Adding of $\mathcal{H}$-matrices $M := M_1 \oplus M_2$
h2h(M_1, M_2,...)	Conversion of an $\mathcal{H}$-matrix M_1 to the $\mathcal{H}$-matrix M_2
h2r(M, R,...)	Conversion an $\mathcal{H}$-matrix to an rank-k matrix
h2f(M, F,...)	Conversion an $\mathcal{H}$-matrix to an dense matrix

Table 5.2: The procedures which are applied for adding two $\mathcal{H}$-matrices with different block structures. M, M_i are $\mathcal{H}$-matrices, R, R_i are rank-k matrices, F, F_i are dense matrices, $i = 1, 2$.

5.10 Complexity Estimates

Lemma 5.10.1 *Let I be an index set, $n := |I|$ and T_I a balanced, binary cluster tree and $T := T_{I\times I}$ a block cluster tree. Let the depth of the tree is $p(T) = \mathcal{O}(\log n)$. Then the number of clusters on the level i is $|T_I^{(i)}| = 2^i$ for $0 \le i \le p(T)$ and $|V(T_I)| = 2|T| - 1 = \mathcal{O}(n)$.*

Proof: see [27].

Theorem 5.10.1 *Let $n := \max\{|I|, |J|, |K|\}$, $m := |J|$, $n \geq m$. Let $T := T_{I\times J}$, $T_{I\times K}$, $T_{K\times J}$ be block cluster trees, $depth(T) = \log n$, $M \in \mathcal{H}(T, k)$, $k := max\{k, n_{min}\}$. The storage requirement and the computational costs of $\mathcal{H}$-matrix operations are given in Table 5.3.*

Operation	Description	Complexity
$storage(M)$	$M \in \mathcal{H}(T_{I\times J}, k)$	$\mathcal{O}(C_{sp}kn \log n$
Mx	$M \in \mathcal{H}(T_{I\times J}, k), x \in \mathbb{R}^{\lvert J\rvert}$	$\mathcal{O}(C_{sp}kn \log n)$
$M' \oplus M''$	$M', M'' \in \mathcal{H}(T_{I\times J}, k)$	$\mathcal{O}(C_{sp}k^2 n \log n)$
$M' \odot M''$	$M' \in \mathcal{H}(T_{I\times K}, k)$, $M'' \in \mathcal{H}(T_{K\times J}, k)$	$\mathcal{O}(C_{sp}^3 k^2 n \log^2 n)$
M^{-1}	$M \in \mathcal{H}(T_{I\times I}, k)$	$\mathcal{O}(C_{sp}^3 k^2 n \log^2 n)$
$\mathcal{H}$-LU	$M \in \mathcal{H}(T_{I\times I}, k)$	$\mathcal{O}(C_{sp}^3 k^2 n \log^2 n)$
$M \oplus R$	$M \in \mathcal{H}(T_{I\times J}, k)$, $R \in \mathcal{R}(k, I, J)$	$\mathcal{O}(k^2(n + m))$

Table 5.3: The costs of $\mathcal{H}$-matrix arithmetical operations, $n := \max\{|I|, |J|, |K|\}$.

Proof: See Lemma 5.13 for the MV multiplication and the MM addition. See Lemmas 2.10, 2.17, 2.19 in [29] for the MM multiplication. See also [33], [27].

Lemma 5.10.2 *Let $T := T_{I\times J}$ be a block cluster tree. The cost of the multiplication of a matrix $M \in \mathcal{H}(T, k)$ and a vector $v \in \mathbb{R}^{|J|}$ is*

$$N_{\mathcal{H}\cdot v}(T, k) \leq 2N_{St}(T, k), \tag{5.29}$$

where $N_{St}(T, k)$ is the storage requirement for M.

Proof: The scalar product of two vectors $u, v \in \mathbb{R}^m$ costs $m + (m - 1) < 2m$ (there are m multiplications and $m-1$ additions). The storage of a rank-k matrix $R = AB^T$ is $N_{St,R}(k, n, m) = k(n+m)$ and the cost of $R\cdot v$ is $2k(n-1)+2k(m-1) \leq 2k(n+m)$. The storage requirement of a dense matrix $F \in \mathbb{R}^{n\times m}$ is $n \cdot m$ and the computational cost of $F \cdot v$ is $(2m - 1) \cdot n \leq 2n \cdot m$. Summing the costs for all admissible blocks we prove the lemma.

Lemma 5.10.3 *Let $T := T_{I\times J}$, $n := |I| \geq |J|$. The removal of i-th row from a matrix $M \in \mathcal{H}(T, k)$ costs $2C_{sp}kn$.*

Proof: This removal procedure updates the old data structure. Let $p := depth(T_{I\times J})$. algorithm goes through the whole tree $T_{I\times J}$ and removes i-th row from blocks $\{(t, s) \in \mathcal{L}(T), i \in t\}$. Therefore, the complexity is

$$\sum_{l=0}^{p} \sum_{(t,s)\in\mathcal{L}(T,l), i\in t} (|t| - 1)kC_{sp} \overset{|t|=2^{p-l}}{\leq} C_{sp}k \sum_{l=0}^{p} (2^{p-l} - 1) \tag{5.30}$$

$$\leq C_{sp}k\sum_{l=0}^{p}2^l \leq 2C_{sp}kn. \tag{5.31}$$

Similarly, the cost of the removal of a column is bounded by $2C_{sp}kn$.

Lemma 5.10.4 *Let T_I be a cluster tree. Adding a rank-1 matrix $R \in \mathcal{R}(1, I, I)$ to the hierarchical matrix $M \in \mathcal{H}(T_{I\times I}, k)$ costs $\mathcal{O}(k^2 n)$, where $n = |I|$.*

Proof: Let $p := depth(T_{I\times I})$. We know that the truncated addition of two rank-k matrices $M|_{t\times s}$ and $R|_{t\times s}$ costs $\mathcal{O}(k^2(|t| + |s|))$. Thus, the complexity is

$$N = \sum_{t\in T_I}\mathcal{O}(k^2|t|) = \mathcal{O}(k^2|I|). \tag{5.32}$$

Lemma 5.10.5 *Extracting a column j from a matrix $M \in \mathcal{H}(T_{I\times J}, k)$ costs $\mathcal{O}(k|I|)$.*

Proof: Let $n := |I| \geq |J|$. Extracting a column j from M is equivalent to the multiplication $M \cdot e_j$, $e_j = (0, 0, ...0, 1, 0..., 0)$ and costs $\mathcal{O}(kn \log n)$. But it can be done with the complexity $\mathcal{O}(kn)$. Let T_J be a cluster tree, $\mathcal{L}(T_J)$ the set of all leaves of T_J and s_0 the biggest cluster, such that $j \in s_0 \in T_J$. Denote the set of all successors of s_0 by $S'(s_0)$. Let

$$\mathcal{L}'(T_{I\times J}) := \{(t, s) \in \mathcal{L}(T_{I\times J})|t \in T_I, \quad s \in T_J \text{ and } s \in S'(s_0)\}. \tag{5.33}$$

Then the complexity is

$$N = \sum_{t\times s\in\mathcal{L}'(T_{I\times J}),} k(|t| + |s|) \leq \sum_{t\in\mathcal{L}(T_I)} 2k|t| = \mathcal{O}(k|I|). \tag{5.34}$$

Similarly, the extraction of a row costs $\mathcal{O}(k|J|)$.

Lemma 5.10.6 *Let $M \in \mathcal{H}(T_{I\times J}, k)$, $I' \subseteq I$, $J' \subseteq J$, $n' = \max\{|I'|, |J'|\}$. The hierarchical conversion (see (5.21)) of $M|_{I'\times J'}$ to $R \in \mathcal{R}(k, I', J')$ is of complexity*

$$N \leq \mathcal{O}(k^2 qn' \log n), \text{ where } q := \max_l \sum_{\substack{(I'\times J')\cap(t\times s)\neq\varnothing \\ (t\times s)\in T^{(l)}_{I\times J}}} 1. \tag{5.35}$$

Proof: Here q is the maximal number of subblocks on the same level which have to be summed. The conversion (see Section 5.9.3) at one level is of complexity $\mathcal{O}(k^2 qn')$. There are $\log n$ levels and the total complexity is $N \leq \mathcal{O}(k^2 qn' \log n)$.

Definition 5.10.1 *Let $I' \subseteq I$, $J' \subseteq J$ be index sets, $T := T_{I\times J}$, $T' := T'_{I'\times J'}$ block cluster trees. Let q be the maximal number of subblocks which form a block $(t' \times s') \in \mathcal{L}(T'_{I'\times J'})$. The value q is defined as following*

$$q := \max_{(t\times s)\in\mathcal{L}(T)} \sum_{\substack{(t\times s)\cap(t'\times s')\neq\varnothing \\ (t'\times s')\in\mathcal{L}(T')}} 1. \tag{5.36}$$

Lemma 5.10.7 *Let $I' \subseteq I$, $J' \subseteq J$ be index sets, $T := T_{I\times J}$, $T' := T_{I'\times J'}$ the block cluster trees, $p := depth(T)$, $M \in \mathcal{H}(T,k)$ and $M' \in \mathcal{H}(T',k)$. The complexity of computing $M := M'|^{I\times J}$ is*

$$N = \mathcal{O}(k^2 qn \log n \log n'),$$

where $n := \max\{|I|,|J|\}$, $n' := \max\{|I'|,|J'|\}$, $depth(T') = \log n'$ and q is defined in (5.36).

Proof: Suppose that for each leaf $b = (t\times s) \in \mathcal{L}(T)$ there are $q(b)$ subblocks in T' which contain elements from $M|_b$. The search of $q(b)$ subblocks costs in the worst case $q(b)\cdot depth(T') = q(b)\log n_1 \leq q\log n_1$.
By Lemma 5.10.6 the cost of the hierarchical conversion $R|_{t\times s} = \mathcal{T}^{\mathcal{R}\leftarrow\mathcal{H}}(M)$ is $C_1 qk^2(|t|+|s|)\log n'$.
Using Lemma 5.10.6 and decomposing all leaves of T on admissible and inadmissible leaves, we obtain

$$\begin{aligned}
N &\leq \sum_{(t\times s)\in\mathcal{L}^+(T)} Cqk^2(|t|+|s|)\log n' + \sum_{(t\times s)\in\mathcal{L}^-(T)} C_1(|t|\cdot|s|) \\
&\leq \sum_{(t\times s)\in\mathcal{L}^+(T)} Cqk^2(|t|+|s|)\log n' + \sum_{(t\times s)\in\mathcal{L}^-(T)} C_1 n_{min}(|t|+|s|) \\
&\leq \sum_{i=0}^{p} \sum_{(t\times s)\in T^{(i)}} max\{Cqk^2\log n', C_1 n_{min}\}(|t|+|s|) \\
&\overset{Cqk^2\log n' > C_1 n_{min}}{=} Cqk^2\log n'\Big(\sum_{i=0}^{p}\sum_{(t\times s)\in T^{(i)}}|t| + \sum_{i=0}^{p}\sum_{(t\times s)\in T^{(i)}}|s|\Big) \\
&\leq 2Cqk^2\log n' \sum_{i=0}^{p}\sum_{t\in T^{(i)}}|t| \\
&\leq 2Cqk^2\log n' \sum_{i=0}^{p}|I| \\
&= 2Cqk^2(p+1)\log n'|I| = \mathcal{O}(k^2 qn\log n\log n').
\end{aligned}$$

■

Lemma 5.10.8 *Let $I' \subseteq I$, $I'' \subseteq I$, $J' \subseteq J$, $J'' \subseteq J$ be index sets, $T := T_{I\times J}$, $T' := T_{I'\times J'}$, $T'' := T_{I''\times J''}$ the block cluster trees, $p := depth(T)$, $n := \max\{|I|,|J|\}$.*

Let $M \in \mathcal{H}(T,k)$, $M_1 \in \mathcal{H}(T',k)$ and $M_2 \in \mathcal{H}(T'',k)$ two $\mathcal{H}$-matrices with different block cluster trees. The complexity of computing

$$M = M_1|^{I\times J} \oplus M_2|^{I\times J}$$

is $\mathcal{O}(qk^2 n \log^2 n)$, where q is defined in (5.36).

Proof: Let $depth(T') = \log n_1$ and $depth(T'') = \log n_2$. From Lemma 5.10.7 follows that the cost of computing $M' := M_1|^{I\times J}$ and $M'' := M_2|^{I\times J}$ is

$$\mathcal{O}(q(\log n_1 + \log n_2)k^2 n \log n).$$

The cost of adding $M' \oplus_k M''$ is $\mathcal{O}(k^2 n \log n)$. Taken into account that $n_1 \simeq \frac{n}{2}$ and $n_2 \simeq \frac{n}{2}$ the total cost is

$$\mathcal{O}(q(\log n_1 + \log n_2)k^2 n \log n) + \mathcal{O}(k^2 n \log n) \leq \mathcal{O}(qk^2 n \log^2 n).$$

6 Application of $\mathcal{H}$-matrices to HDD

The HDD method from Chapter 4 requires the exact matrix arithmetic, which is expensive. To avoid it we approximate all matrices by the corresponding $\mathcal{H}$-matrices and then apply the efficient $\mathcal{H}$-matrix technique. Thus, we reduce the computational cost and storage requirements extremely.

In this section we explain how to apply $\mathcal{H}$-matrices for the approximation of Ψ_ω and Φ_ω. We explain the algorithm "Leaves to Root" in terms of the Schur complements defined on ω_1, ω_2 and ω.

6.1 Notation and Algorithm

In Table 6.1 we recall four types of mappings which are present in the HDD method. All of them will be approximated in the $\mathcal{H}$-matrix format.

Name	Mapping
domain-to-boundary	$\Psi^f_\omega : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\partial\omega)}$
boundary-to-boundary	$\Psi^g_\omega : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\partial\omega)}$
domain-to-interface	$\Phi^f_\omega : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\gamma)}$
boundary-to-interface	$\Phi^g_\omega : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\gamma)}$

Table 6.1: Four mappings, which are used in HDD. $\omega \in T_{\mathcal{T}_h}$.

The properties of these mappings are:

1. The mappings Ψ^f_ω and Φ^f_ω are approximated by $\mathcal{H}$-matrices with the standard admissibility condition (see Section 5.5.1).

2. The mapping Ψ^g_ω is approximated by an $\mathcal{H}$-matrix with the weak admissibility condition (see Section 5.5.2).

3. The mapping Φ^g_ω is approximated by a low-rank matrix (as a consequence of using the weak admissibility condition).

4. The algorithm "Leaves to Root" builds the mappings Ψ^g_ω and Ψ^f_ω. The mappings Ψ^g_ω and Ψ^f_ω are computed automatically.

5. If the global stiffness matrix after discretisation of the initial problem (4.1) is self-adjoint, then all matrices Ψ^g_ω, $\omega \in T_{\mathcal{T}_h}$, are positive definite and symmetric.

Notation 6.1.1 *Let us denote an $\mathcal{H}$-matrix approximation of a mapping χ by $(\chi)^{\mathcal{H}}$.*

Later on, for simplicity of the notation, we will omit $()^{\mathcal{H}}$.

Further, we will assume that $\omega = \omega_1 \cup \omega_2$, $\omega, \omega_i \in T_{\mathcal{T}_h}$, $\Gamma_{\omega,i} = \partial\omega \cap \overline{\omega_i}$ and $\gamma = \gamma_\omega = \partial\omega_i \backslash \partial\omega$, $i = 1, 2$.

6.2 Algorithm of Applying $\mathcal{H}$-Matrices

1. We start with computing the systems of linear equations for all leaves of $T_{\mathcal{T}_h}$ as it is done in Section 4.3.1 (leaves of $T_{\mathcal{T}_h}$ are triangles). Note that only in this step we apply formulae from Section 4.3.1. As a result we have a system $A\mathbf{u} = F\mathbf{c}$ for each leaf $\omega \in T_{\mathcal{T}_h}$, where $A \in \mathbb{R}^{3\times 3}$ and $F \in \mathbb{R}^{3\times 3}$.

2. Suppose the systems $A^{(1)}\mathbf{u} = F^{(1)}\mathbf{c}$ and $A^{(2)}\mathbf{u} = F^{(2)}\mathbf{c}$ for subdomains ω_1 and ω_2, respectively, are given. Now, we would like to construct the matrices A and F which appear in the system $A\mathbf{u} = F\mathbf{c}$ for the domain ω. We construct A from the matrices $A^{(1)}$, $A^{(2)}$ and F from $F^{(1)}$ and $F^{(2)}$ as it was shown in Section 4.3.3. "Construct" means that we simply sum the elements which correspond to the common points in ω_1 and ω_2 and copy the elements which correspond to the unique points. As soon as the matrices A and F become large, we compute their $\mathcal{H}$-matrix approximations.

3. Let the system of linear equations $A\mathbf{u} = F\mathbf{c}$ for $\omega \in T_{\mathcal{T}_h}$ be given. A is the stiffness matrix for the domain $\overline{\omega}$ after elimination of the unknowns corresponding to $I(\mathring{\omega} \backslash \gamma_\omega)$. The matrix F comes from the numerical integration. Here $A : \mathbb{R}^{I(\partial\omega\cup\gamma)} \to \mathbb{R}^{I(\partial\omega\cup\gamma)}$, $\mathbf{u} \in \mathbb{R}^{I(\partial\omega\cup\gamma)}$, $F : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\partial\omega\cup\gamma)}$ and $\mathbf{c} \in \mathbb{R}^{I(\omega)}$. Decompose the unknown vector $\mathbf{u}$ into two components $\mathbf{u}_1 \in \mathbb{R}^{I(\partial\omega)}$ and $\mathbf{u}_2 \in \mathbb{R}^{I(\gamma)}$. Then the system of linear equations $A\mathbf{u} = F\mathbf{c}$ takes on form

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} \mathbf{c}, \tag{6.1}$$

where
$A_{11} : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\partial\omega)}$, $A_{12} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\partial\omega)}$, $A_{21} : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\gamma)}$,
$A_{22} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\gamma)}$, $F_1 : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\partial\omega)}$, $F_2 : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\gamma)}$, $\mathbf{u}_1 \in \mathbb{R}^{I(\partial\omega)}$, $\mathbf{u}_2 \in \mathbb{R}^{I(\gamma)}$,
$\mathbf{c} \in \mathbb{R}^{I(\omega)}$.

4. Now we eliminate the unknown vector $\mathbf{u}_2$ as shown in (6.2). We multiply both sides of (6.1) on $A_{12}A_{22}^{-1}$ and subtract the second row from the first row

$$\begin{pmatrix} A_{11} - A_{12}A_{22}^{-1}A_{21} & 0 \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} = \begin{pmatrix} F_1 - A_{12}A_{22}^{-1}F_2 \\ F_2 \end{pmatrix} \mathbf{c}. \tag{6.2}$$

Note that we do not multiply the matrices A_{22}^{-1} and F_2 in (6.2). We rewrite the last system as two equations

$$\begin{gathered} \tilde{A}\mathbf{u}_1 := (A_{11} - A_{12}A_{22}^{-1}A_{21})\mathbf{u}_1 = (F_1 - A_{12}A_{22}^{-1}F_2)\mathbf{c}, \\ A_{22}\mathbf{u}_2 = F_2\mathbf{c} - A_{21}\mathbf{u}_1. \end{gathered} \tag{6.3}$$

The unknown vector $\mathbf{u}_2$ is computed as follows (compare with (4.43))

$$\mathbf{u}_2 = A_{22}^{-1} F_2 \mathbf{c} - A_{22}^{-1} A_{21} \mathbf{u}_1.$$

The explicit expressions for the mappings Ψ_ω and Φ_ω follow from (6.3):

$$\Psi_\omega^g := A_{11} - A_{12} A_{22}^{-1} A_{21}, \tag{6.4}$$

$$\Psi_\omega^f := F_1 - A_{12} A_{22}^{-1} F_2, \tag{6.5}$$

$$\Phi_\omega^g := -A_{22}^{-1} A_{21}, \tag{6.6}$$

$$\Phi_\omega^f := A_{22}^{-1} F_2, \tag{6.7}$$

As soon as the rank of A_{22} is larger than parameter n_{min} (see Section 5.5.2), we apply SVD to convert A and F from the dense matrix format to the $\mathcal{H}$-matrix format. Denote the standard operations $-$ and $\cdot$ in the class of $\mathcal{H}$-matrices by $\ominus$ and $\odot$. After approximation of all matrices in (6.4)-(6.7) by $\mathcal{H}$-matrices we obtain

$$(\Psi_\omega^g)^{\mathcal{H}} := (A_{11})^{\mathcal{H}} \ominus (A_{12})^{\mathcal{H}} \odot (A_{22}^{-1})^{\mathcal{H}} \odot (A_{21})^{\mathcal{H}}, \tag{6.8}$$

$$(\Psi_\omega^f)^{\mathcal{H}} := (F_1)^{\mathcal{H}} \ominus (A_{12})^{\mathcal{H}} \odot (A_{22}^{-1})^{\mathcal{H}} \odot (F_2)^{\mathcal{H}}, \tag{6.9}$$

$$(\Phi_\omega^g)^{\mathcal{H}} := -(A_{22}^{-1})^{\mathcal{H}} \odot (A_{21})^{\mathcal{H}}, \tag{6.10}$$

$$(\Phi_\omega^f)^{\mathcal{H}} := (A_{22}^{-1})^{\mathcal{H}} \odot (F_2)^{\mathcal{H}}. \tag{6.11}$$

Note that we multiply $\mathcal{H}$-matrices only in (6.8) and (6.9). In (6.10) and (6.11) we store the multipliers and later on perform only two times matrix-vector multiplications.

5. We repeat steps 2-5 for all other $\omega \in T_{\mathcal{T}_h}$ and stop when $\omega = \Omega$.

Remark 6.2.1 *The matrices A_{22}^{-1}, F_1, F_2, A_{11} are approximated in the $\mathcal{H}$-matrix format, A_{12} and A_{21} are approximated in the low-rank matrix format. For simplicity of the further notation we omit the superscript $^{\mathcal{H}}$.*

Example 6.2.1 *Examples of the matrices A and F (Ψ_ω^g and Ψ_ω^f) from (6.1) are shown in Figures 6.1 and 6.2. In Fig. 6.1 one can see that off-diagonal blocks are low-rank matrices (grey blocks). In Fig. 6.2 the off-diagonal blocks are $\mathcal{H}$-matrices. The white blocks indicate zero matrices and the dark blocks indicate dense matrices.*

Remark 6.2.2 *The fact that the matrix A is approximated by an $\mathcal{H}$-matrix with weakly admissible blocks plays an important role. From this fact follows that the matrices A_{12} and A_{21} are low-rank matrices and multiplications in $A_{12} \odot A_{22}^{-1}$ and in $A_{22}^{-1} \odot A_{21}$ are therefore quite cheap.*

The following Theorem 6.2.1 (see [47]) is important for the application of the $\mathcal{H}$-matrix technique to HDD. This theorem proves the existence of an $\mathcal{H}$-matrix approximation of the Schur complement.

Notation 6.2.1 *Let t, $s \in T_I$ be two clusters, $T_{I\times I}$ a block cluster tree and $s \times t \in T_{I\times I}$. We denote the restriction of the block cluster tree $T_{I\times I}$ to $s \times t$ by $T|_{s\times t}$*

Assumption 6.2.1 ***(Existence of an $\mathcal{H}$-matrix approx. to the inverse)*** *Let $A \in \mathcal{H}(T_{I\times I}, k)$, $n := |I|$. For any $\varepsilon > 0$ and $r := \{1, ..., n_1\}$, $n_1 \leq n$, the minor $B := A|_{r\times r}$ is invertible and there exists an $\mathcal{H}$-matrix $B_{\mathcal{H}}^{-1} \in \mathcal{H}(T|_{r\times r}, k_{inv})$ with*

$$k_{inv} := (\log^2 n)|\log\varepsilon|^3 \text{ and } \|B^{-1} - B_{\mathcal{H}}^{-1}\|_2 \leq C_{inv}\varepsilon, \quad C_{inv} \in \mathbb{R}_+. \tag{6.12}$$

Proof: See Assumption 2 in [47].

Theorem 6.2.1 ***(Approximation of Schur complements)*** *Let T_I be a cluster tree, $T := T_{I\times I}$ and p be depth of T. Let $A \in \mathcal{H}(T, k_{inv})$ with k_{inv} from (6.12), $b := s \times t \in T$ and $r \in T_I$. Then the Schur complement*

$$S(s,t) = A|_{s\times t} - A|_{s\times r}(A|_{r\times r})^{-1}A|_{r\times t}$$

can be approximated by $S_{\mathcal{H}}(s,t) \in \mathcal{H}(T|_{s\times t}, k')$ where $k' \lesssim (p+1)^2 k_{inv}$, such that

$$\|S(s,t) - S_{\mathcal{H}}(s,t)\|_2 < C_{inv}\|A\|_2^2\varepsilon, \quad C_{inv} \in \mathbb{R}_+.$$

Proof: See Theorem 1 in [47].

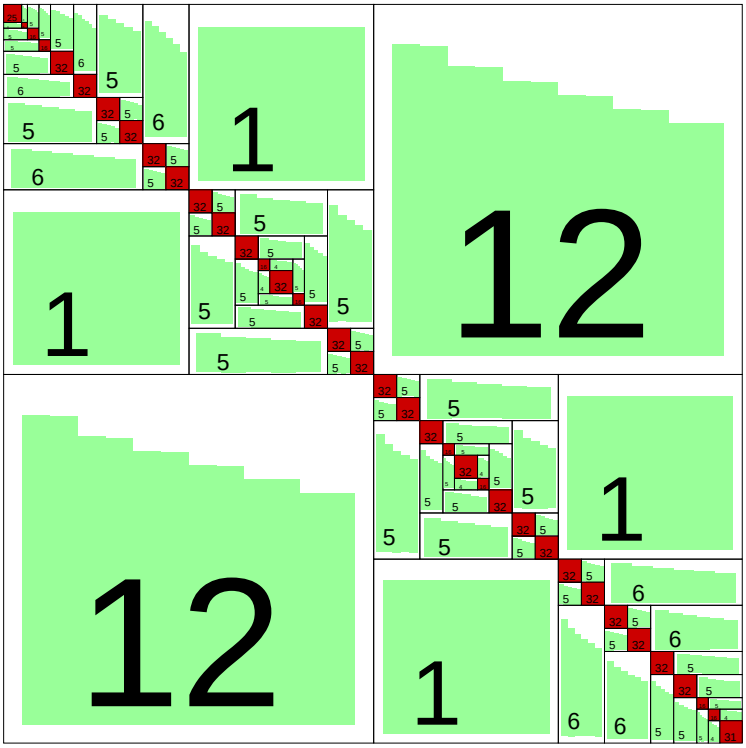


Figure 6.1: An $\mathcal{H}$-matrix approximation to $\Psi_\omega^g \in \mathbb{R}^{I\times I}$, $I := I(\partial\omega)$. The dark blocks are dense matrices and grey blocks are low-rank matrices. The numbers inside the blocks indicate ranks of these blocks.

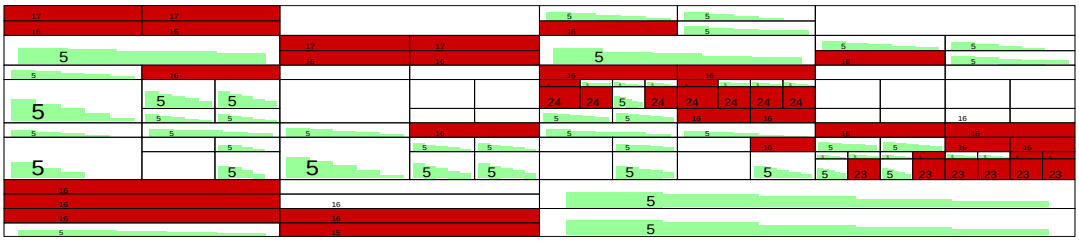


Figure 6.2: An $\mathcal{H}$-matrix approximation to $(\Psi_\omega^f)^{\mathcal{H}} \in \mathbb{R}^{I\times J}$, $I := I(\partial\omega)$, $J := J(\omega)$, $|I| = 256$, $|I| = 4225$. The dark blocks are dense matrices and grey blocks are low-rank matrices. The numbers inside the blocks are ranks of these blocks. The white blocks are zero blocks.

Remark 6.2.3 *In order to build a rank-k approximation of the matrix Φ_ω^g we use the fact that the singular values $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_k \geq \sigma_{k+1} \geq ... \geq \sigma_n$ of Φ_ω^g decay exponentially (see Fig. 6.3). We only consider the k largest singular values $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_k$ (see Section 5.6).*

Recall that for simplicity of the notation we write γ instead of γ_ω.

Lemma 6.2.1 *We denote the Schur complement $A_{11} - A_{12}A_{22}^{-1}A_{21}$ by S_ω, where $A_{11} \in \mathcal{H}(T_{I(\partial\omega)\times I(\partial\omega)}, k)$, $A_{12} \in \mathcal{R}(I(\partial\omega), I(\gamma), k)$, $A_{21} \in \mathcal{R}(I(\gamma), I(\partial\omega), k)$, $A_{22} \in \mathcal{H}(T_{I(\gamma)\times I(\gamma)}, k)$. Let $n_{h,\gamma} := |I(\gamma)|$. Let the model domain be as in Remark 7.1.2. The computation of S_ω, $\omega \in T_{\mathcal{T}_h}$, costs*

$$N(S_\omega) \leq Ck^2 n_{h,\gamma} \log^2 n_{h,\gamma}, \quad C \in \mathbb{R}_+. \tag{6.13}$$

Proof: Due to Table 5.3 the complexity of the inversion A_{22}^{-1} is $C_1 k^2 n_{h,\gamma} \log^2 n_{h,\gamma}$, $C_1 \in \mathbb{R}_+$. The complexity of the multiplication $A_{12} \cdot A_{22}^{-1}$ is k-times the MV multiplication, i.e., $C_2 k^2 n_{h,\gamma} \log n_{h,\gamma}$, $C_2 \in \mathbb{R}_+$.
We assume that for the domain in Remark 7.1.2 $|I(\partial\omega)| \leq 6 \cdot |I(\gamma)|$ hold. The complexity of the multiplication $(A_{12}A_{22}^{-1}) \cdot A_{21}$ (product of two low-rank matrices) is

$$2k^2(|I(\partial\omega)| + |I(\gamma)|) + \mathcal{O}(k^3) \leq 14k^2 n_{h,\gamma} + \mathcal{O}(k^3).$$

The complexity of the subtraction $A_{11} - A_{12}A_{22}^{-1} \cdot A_{21}$ is

$$C_3 k^2 |I(\partial\omega)| \log |I(\partial\omega)| \leq C_4 k^2 n_{h,\gamma} \log n_{h,\gamma}, \quad C_3, C_4 \in \mathbb{R}_+.$$

Thus,

$$N(S_\omega) \leq (C_1{+}C_2{+}C_4)k^2 n_{h,\gamma} \log n_{h,\gamma} {+} 14k^2 n_{h,\gamma} {+} \mathcal{O}(k^3) \leq Ck^2 n_{h,\gamma} \log^2 n_{h,\gamma}, \quad C \in \mathbb{R}_+. \tag{6.14}$$

■

Algorithm 6.2.1 *(Elimination of u_i, $i \in I(\gamma)$)*
elimination*($\mathcal{H}$-matrix M, index set $I(\gamma)$)*
begin
$M_{11} := M[0]$;
$M_{21} := M[1]$;
$M_{12} := M[2]$;
$M_{22} := M[3]$; /* *Corresponds to* $I(\gamma)$ */
$\tilde{M}_{11} := M_{11} \ominus M_{12} \odot M_{22}^{-1} \odot M_{21}$;
return $\tilde{M}_{11}$;
end;

Lemma 6.2.2 *Let $Z_\omega := F_1 - A_{12}A_{22}^{-1}F_2$, where $A_{12} \in \mathcal{R}(I(\partial\omega), I(\gamma), k)$, $F_1 \in \mathcal{H}(T_{I(\partial\omega)\times I(\omega)}, k)$, $F_2 \in \mathcal{H}(T_{I(\gamma)\times I(\omega)}, k)$, $A_{22} \in \mathcal{H}(T_{I(\gamma)\times I(\gamma)}, k)$, $n_{h,\gamma} := |I(\gamma)|$ and $n_h := |I(\omega)|$. Let the model domain be as in Remark 7.1.2. The computation of Z_ω, $\omega \in T_{\mathcal{T}_h}$, costs*

$$N(Z_\omega) \leq Ck^2 n_h \log^2 n_h, \quad C \in \mathbb{R}_+. \tag{6.15}$$

Proof: The complexity of the multiplication $A_{12} \cdot A_{22}^{-1}$ is k-times the MV multiplication, i.e., $C_1 k^2 n_{h,\gamma} \log n_{h,\gamma}$, $C_1 \in \mathbb{R}_+$. The complexity of the multiplication $(A_{12}A_{22}^{-1}) \cdot F_2$ (the product of a low-rank matrix and an $\mathcal{H}$-matrix) is equal to $C_2 k^2 n_h \log n_h$, $C_2 \in \mathbb{R}_+$. The complexity of the subtraction $F_1 - A_{12}A_{22}^{-1}F_2$ is $C_3 k^2 n_h \log n_h$, $C_3 \in \mathbb{R}_+$. Thus,

$$N(Z_\omega) = C_1 k^2 n_{h,\gamma} \log n_{h,\gamma} + C_2 k^2 n_h \log n_h + C_3 k^2 n_h \log n_h \leq Ck^2 n_h \log n_h, \; C \in \mathbb{R}_+.$$

■

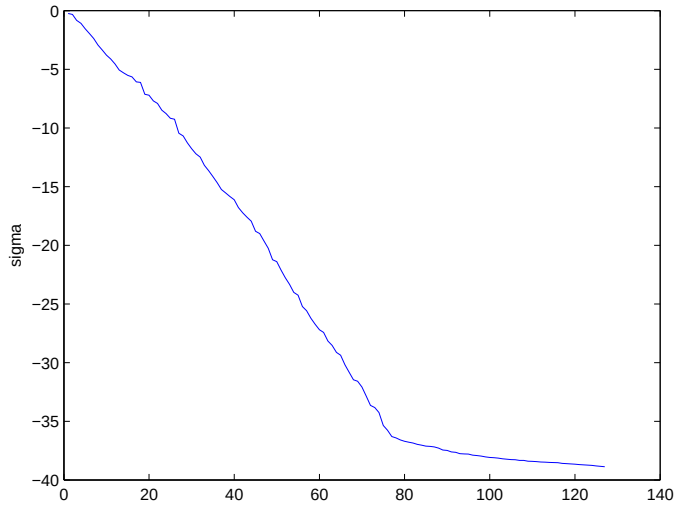


Figure 6.3: Exponential decay of the singular values σ_i of the matrix $(\Phi_\omega^g)^{\mathcal{H}}$ in a log scale. The index i is shown on the horizontal axis and the singular values σ_i on the vertical logarithmic axis, e.g., $\sigma_{20} \approx 10^{-7}$.

In Table 6.2 we compute the singular values of the matrix Φ^f_ω, $\omega = \Omega$, with 129^2 degrees of freedom. One can see that, in contrast to Φ^g_ω, the singular values do not decay exponentially.

σ_1	$3.48 * 10^{-4}$
σ_{10}	$1.27 * 10^{-4}$
σ_{20}	$9.24 * 10^{-5}$
σ_{50}	$6.84 * 10^{-5}$

Table 6.2: A very slow decay of singular values σ_i of $(\Phi^f_\omega)^{\mathcal{H}}$.

Remark 6.2.4 *This very slow decay of singular values of Φ^f_ω results in a very large rank k in the rank-k approximation. By this reason Φ^f_ω is approximate in the class of $\mathcal{H}$-matrices.*

6.3 Hierarchical Construction on Incompatible Index Sets

This Section contains technical details about $\mathcal{H}$-matrix constructions of $(\Psi^g_\omega)^{\mathcal{H}}$ from $(\Psi^g_{\omega_1})^{\mathcal{H}}$, $(\Psi^g_{\omega_2})^{\mathcal{H}}$ and $(\Psi^f_\omega)^{\mathcal{H}}$ from $(\Psi^f_{\omega_1})^{\mathcal{H}}$, $(\Psi^f_{\omega_2})^{\mathcal{H}}$.

Let $T_{\mathcal{T}_h}$ be a domain decomposition tree (the root has level 0). Let us suppose that we have a domain ω on level $l-1$ with two sons ω_1 and ω_2 on level l. In the next two subsections we show the efficient constructions of $(\Psi^g_\omega)^{\mathcal{H}}$ from $(\Psi^g_{\omega_1})^{\mathcal{H}}$, $(\Psi^g_{\omega_2})^{\mathcal{H}}$ and $(\Psi^f_\omega)^{\mathcal{H}}$ from $(\Psi^f_{\omega_1})^{\mathcal{H}}$, $(\Psi^f_{\omega_2})^{\mathcal{H}}$.
Let us change the notation:

$$I(\Gamma_1) := I(\Gamma_{\omega,1}) = I(\partial\omega \cap \omega_1),$$

$$I(\Gamma_2) := I(\Gamma_{\omega,2}) = I(\partial\omega \cap \omega_2).$$

Now, we have the following decompositions:

$$I(\partial\omega_1) = I(\Gamma_1) \cup I(\gamma) \text{ and } I(\partial\omega_2) = I(\Gamma_2) \cup I(\gamma).$$

6.3.1 Building $(\Psi^g_\omega)^{\mathcal{H}}$ from $(\Psi^g_{\omega_1})^{\mathcal{H}}$ and $(\Psi^g_{\omega_2})^{\mathcal{H}}$

Let $\tilde{H} \in \mathcal{H}(T_{I(\partial\omega\cup\gamma)\times I(\partial\omega\cup\gamma)}, k)$. Define the following matrices:

$$H_1 := (\Psi^g_{\omega_1})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_1)\times I(\partial\omega_1)}, k), \quad H_2 := (\Psi^g_{\omega_2})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_2)\times I(\partial\omega_2)}, k), \quad (6.16)$$

$$H := (\Psi^g_\omega)^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega)\times I(\partial\omega)}, k), \quad (6.17)$$

where $I(\partial\omega\cup\gamma) = I(\partial\omega_1\cup\partial\omega_2)$ (see Remark 4.3.3 and Figures 6.4, 6.5). We want to construct the matrix H from H_1 and H_2. First, we build a new cluster tree $T_{I(\partial\omega\cup\gamma)}$ from the clusters $T_{I(\partial\omega)}$ and $T_{I(\gamma)}$. There are many variants of how to build it, but we want such a cluster tree, which makes it easier to eliminate the unknowns x_i, $i \in I(\gamma)$, i.e, one of the sons of the cluster $I(\partial\omega \cup \gamma)$ should coincide with the index set $I(\gamma)$. As soon as the cluster tree $T_{I(\partial\omega\cup\gamma)}$ is built, we build the block cluster tree $T_{I(\partial\omega\cup\gamma)\times I(\partial\omega\cup\gamma)}$. The block cluster tree $T_{I(\partial\omega\cup\gamma)\times I(\partial\omega\cup\gamma)}$ defines the block structure of $\tilde{H}$. We consider two variants of the block structures:

$$I(\Gamma_i) \times I(\Gamma_i), I(\gamma) \times I(\gamma) \in T_{I(\partial\omega_i)\times I(\partial\omega_i)}, i = 1, 2. \quad (6.18)$$

$$I(\Gamma_i) \times I(\Gamma_i) \notin T_{I(\partial\omega_i)\times I(\partial\omega_i)} \text{ or } I(\gamma) \times I(\gamma) \notin T_{I(\partial\omega_i)\times I(\partial\omega_i)}, i = 1, 2. \quad (6.19)$$

Building algorithm in case (6.18):
Let H_1 and H_2 be defined as in (6.16) and H as in (6.17).

Algorithm 6.3.1 *Building* $H := (\Psi^g_\omega)^{\mathcal{H}}$ *from* $H_1 := (\Psi^g_{\omega_1})^{\mathcal{H}}$ *and* $H_2 := (\Psi^g_{\omega_2})^{\mathcal{H}}$

build_ Ψ^g*(*H_1*,* H_2*,...)*

begin

allocate memory for $\tilde{H}$*;*

$\tilde{H}|_{I(\Gamma_1)\times I(\Gamma_1)} := H_1|_{I(\Gamma_1)\times I(\Gamma_1)}$*;*

$\tilde{H}|_{I(\Gamma_2)\times I(\Gamma_2)} := H_2|_{I(\Gamma_2)\times I(\Gamma_2)}$*;*

$\tilde{H}|_{I(\Gamma_1)\times I(\Gamma_2)} := 0$*;*

$\tilde{H}|_{I(\Gamma_2)\times I(\Gamma_1)} := 0$*;*

/ in Fig. 6.4 denoted by* $d + h$ **/*

$\tilde{H}|_{I(\gamma)\times I(\gamma)} := H_1|_{I(\gamma)\times I(\gamma)} \oplus H_2|_{I(\gamma)\times I(\gamma)}$*;*

$\tilde{H}|_{I(\gamma)\times I(\Gamma_1)\cup I(\Gamma_2)} := (H_1|_{I(\gamma)\times I(\Gamma_1)} \oplus H_2|_{I(\gamma)\times I(\Gamma_2)})$*; /* Sum of two low-rank matrices */*

/ in Fig. 6.4 denoted by* $b + f$ **/*

$\tilde{H}|_{I(\Gamma_1)\cup I(\Gamma_2)\times I(\gamma)} := H_1|_{I(\Gamma_1)\times I(\gamma)} \oplus H_2|_{I(\Gamma_2)\times I(\gamma)}$*; /* Sum of two low-rank matrices */*

$\tilde{H}$*:=extract_rows(*$\tilde{H}$*,* r_1*,* r_2*,* i_1*,* i_2*); /* The output is* r_1*,* r_2 **/*

$\tilde{H}$*:=extract_columns(*$\tilde{H}$*,* c_1*,* c_2*,* j_1*,* j_2*); /* The output is* c_1*,* c_2 **/*

$\tilde{H}$*:=add_rows(*$\tilde{H}$*,* r_1*,* r_2*,* i_3*,* i_4*);*

$\tilde{H}$*:=add_columns(*$\tilde{H}$*,* c_1*,* c_2*,* j_3*,* j_4*);*

H*:=elimination(*$\tilde{H}$*,* $I(\gamma)$*); /* see Algorithm 6.2.1*/*

return H*;*

end*;*

Remark 6.3.1 *Since* $I(\Gamma_1)\cap I(\Gamma_2) = I(\{x, y\}) \neq \varnothing$*, we should remove two repeated columns and two repeated rows from* $\tilde{H}$*. The indices* i_1*,* i_2*,* j_1*,* j_2 *indicate the positions of two rows and two columns which have to be extracted.* i_3*,* i_4 *and* j_3*,* j_4 *are positions of two rows and two columns to which the removed rows and columns* r_1*,* r_2*,* c_1 *and* c_2 *have to be added.*

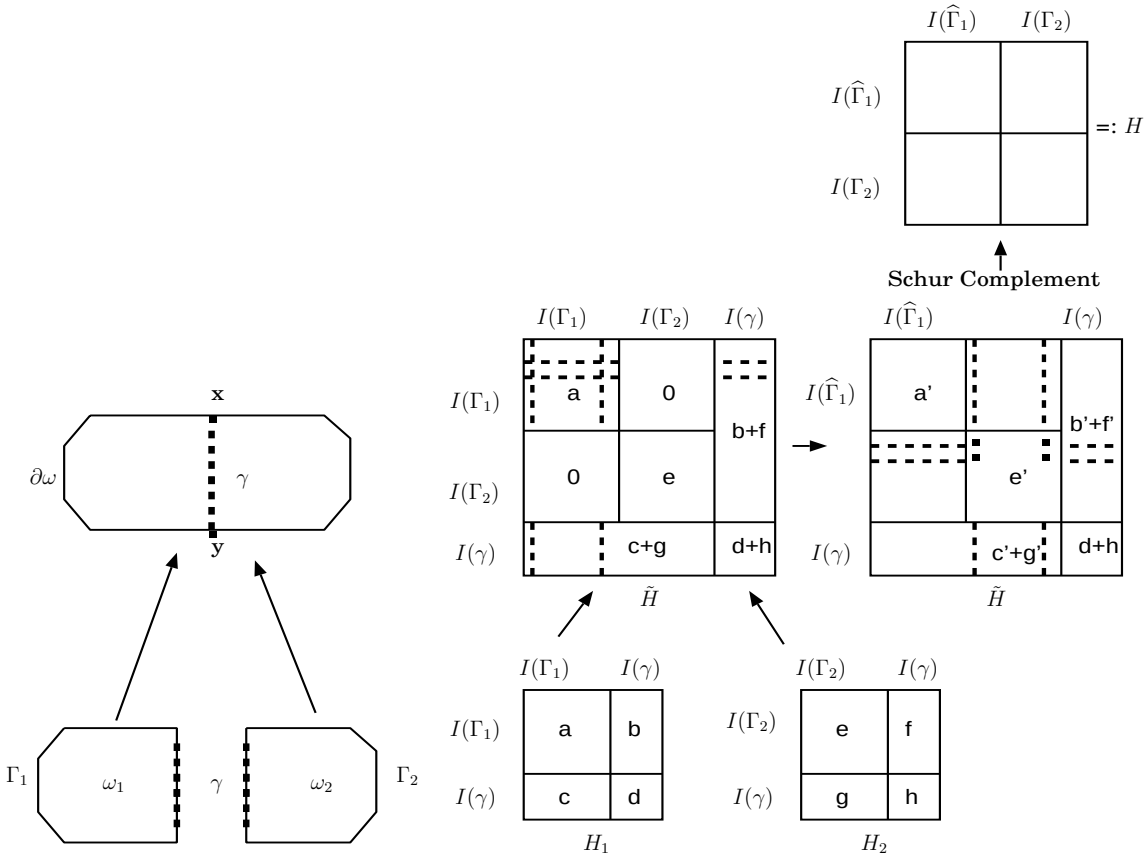


Figure 6.4: Building $H := (\Psi_\omega^g)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega)\times I(\partial\omega)}$ from $H_1 := (\Psi_{\omega_1}^g)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_1)\times I(\partial\omega_1)}$ and $H_2 := (\Psi_{\omega_2}^g)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_2)\times I(\partial\omega_2)}$, where $I(\partial\omega_i) = I(\Gamma_i) \cup I(\gamma)$, $i = 1, 2$, $I(\partial\omega) = I(\widehat{\Gamma}_1) \cup I(\Gamma_2)$, x, y are two common points and $I(\widehat{\Gamma}_1) := I(\Gamma_1) \setminus I(\{x, y\})$. The small letters show the appearance of blocks in different matrices. The dotted lines in $\tilde{H}$ present 2 rows and 2 columns.

Remark 6.3.2 *Figure 6.4 illustrates Algorithm 6.3.1. The first step is the construction of $\tilde{H}$ and then, according to (6.4),*

$$(\Psi_\omega^g)^{\mathcal{H}} = H_{11} \ominus H_{12} \odot H_{22}^{-1} \odot H_{21},$$

with $H_{11} := \tilde{H}|_{I(\partial\omega)\times I(\partial\omega)}$, $H_{12} := \tilde{H}|_{I(\partial\omega)\times I(\gamma)}$,
$H_{21} := \tilde{H}|_{I(\gamma)\times I(\partial\omega)}$, $H_{22} := \tilde{H}|_{I(\gamma)\times I(\gamma)}$.

Lemma 6.3.1 *The cost of building the $\mathcal{H}$-matrix H (see (6.17)) from the $\mathcal{H}$-matrices H_1 and H_2 (see (6.16)) in case (6.18) by Algorithm 6.3.1 is*

$$N \leq Ck^2 n_\gamma \log n_\gamma, \quad \text{where } n_\gamma = |I(\gamma)|, \quad C \in \mathbb{R}_+.$$

Proof: We follow Algorithm 6.3.1 (see the scheme in Fig. 6.4). Let $n_1 := |I(\partial\omega_1)|$, $n_2 := |I(\partial\omega_2)|$, $n := |I(\partial\omega)|$.

1. From Table 5.3 follows that the complexity of adding $H_1|_{I(\gamma)\times I(\gamma)}$ and $H_2|_{I(\gamma)\times I(\gamma)}$ (in Fig. 6.4 denoted by $d + h$) is $\mathcal{O}(k^2 n_\gamma \log n_\gamma)$.

2. From Table 5.1 follows that the complexity of adding the low-rank matrices $H_1|_{I(\gamma)\times I(\Gamma_1)}$ and $H_2|_{I(\gamma)\times I(\Gamma_2)}$ as well as $H_1|_{I(\Gamma_1)\times I(\gamma)}$ and $H_2|_{I(\Gamma_2)\times I(\gamma)}$ is $\mathcal{O}(k^2|I(\partial\omega)|) = \mathcal{O}(k^2 n)$ (see Table 5.1).

3. The removal of two columns and two rows from H_1 by Lemma 5.10.3 costs $\mathcal{O}(k \log n_1)$.

4. Adding four rank-1 matrices to $\tilde{H}$ by Lemma 5.10.4 costs $\mathcal{O}(k^2 n_2)$.

5. Adding four elements to the matrix $\tilde{H}|_{I(\Gamma_2)\times I(\Gamma_2)}$ in general by Lemma 5.10.4 costs $\mathcal{O}(k^2|I(\Gamma_2)|)$. In our case these elements belong to diagonal blocks and should be added to dense matrices. The cost is $\mathcal{O}(1)$.

6. The cost of computing the Schur complement by Lemma 6.2.1 is $\mathcal{O}(k^2 n_\gamma \log^2 n_\gamma)$.

Since the complexity of adding two $\mathcal{H}$-matrices dominates, the total complexity is

$$N \leq Ck^2 n_\gamma \log n_\gamma, \quad C \in \mathbb{R}_+.$$

■

Building algorithm in case (6.19):
Let H_1 and H_2 be defined as in (6.16), H as in (6.17) and $I := I(\partial\omega)$.

Algorithm 6.3.2 *Building $H := (\Psi^g_\omega)^{\mathcal{H}}$ from $H_1 := (\Psi^g_{\omega_1})^{\mathcal{H}}$ and $H_2 := (\Psi^g_{\omega_2})^{\mathcal{H}}$*

```
build_Ψ^g(H_1, H_2,...)
begin
    allocate memory for H̃, H;
    H′:=copy_block_structure(H);
    H″:=copy_block_structure(H);
    h2h(H_1, H′,...);/* Convert H_1 to H′ */
    h2h(H_2, H″,...);/* Convert H_2 to H″ */
    H̃ := H″ ⊕ H′; /* See (6.17) */
    H:=elimination(H̃, I(γ_12)); /* see Algorithm 6.2.1 */
    return H;
end;
```

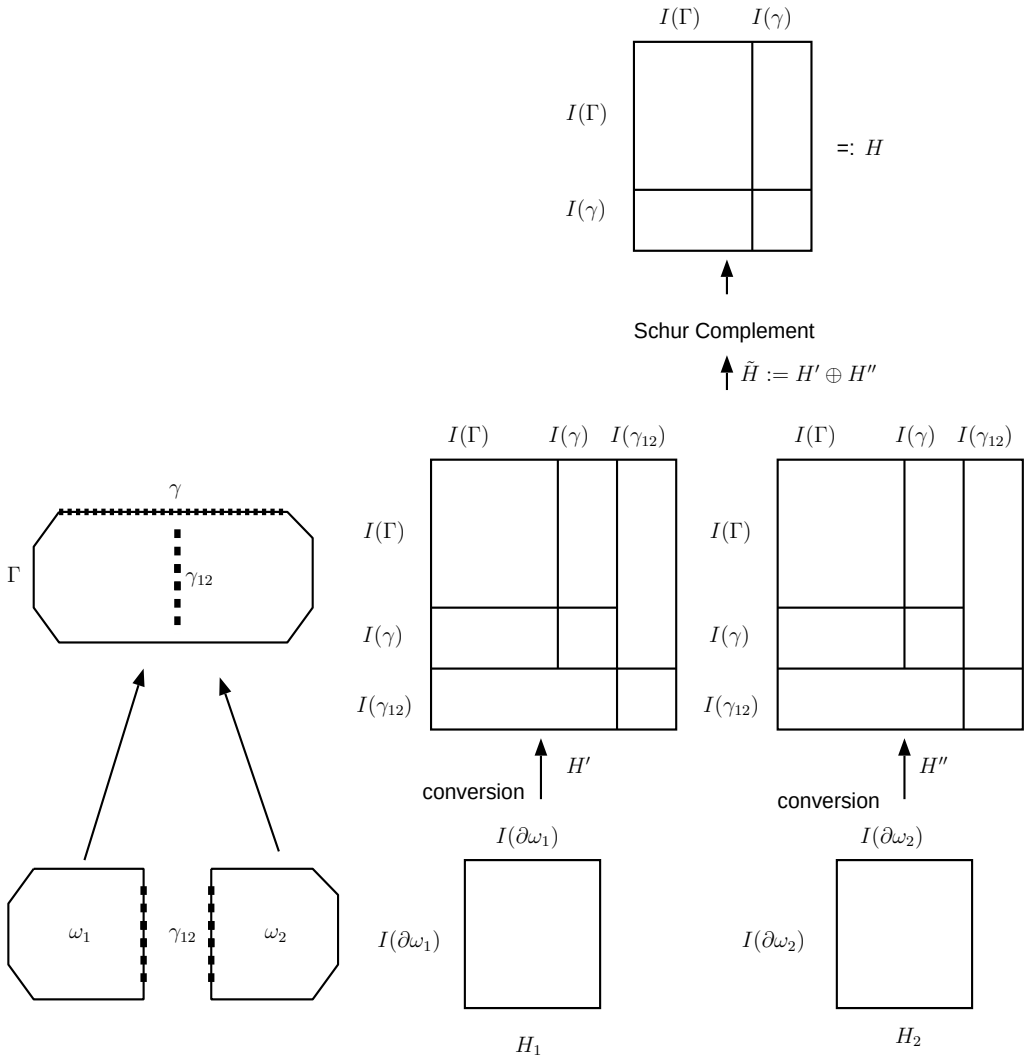


Figure 6.5: Building $H := (\Psi_\omega^g)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega)\times I(\partial\omega)}$ from $H_1 := (\Psi_{\omega_1}^g)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_1)\times I(\partial\omega_1)}$ and $H_2 := (\Psi_{\omega_2}^g)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_2)\times I(\partial\omega_2)}$, $I(\partial\omega_i) = I(\Gamma_i) \cup I(\gamma_{12})$, $i = 1, 2$, $I(\partial\omega) = I(\Gamma) \cup I(\gamma)$. We obtain H after computing $H' \oplus H''$ and the Schur complement, where $H' := H_1|^{I(\partial\omega\cup\gamma_{12})\times I(\partial\omega\cup\gamma_{12})}$, $H'' := H_2|^{I(\partial\omega\cup\gamma_{12})\times I(\partial\omega\cup\gamma_{12})}$.

Remark 6.3.3 *Figure 6.5 illustrates Algorithm 6.3.2. First, it shows how $\tilde{H}$ is constructed. Here*

$$H' := H_1|^{I(\partial\omega\cup\gamma_{12})\times I(\partial\omega\cup\gamma_{12})}, \quad H'' := H_2|^{I(\partial\omega\cup\gamma_{12})\times I(\partial\omega\cup\gamma_{12})}, \quad \text{and } \tilde{H} = H' \oplus H''$$

with $\gamma_{12} := \partial\omega_1 \setminus \partial\omega$. Then, according to (6.4),

$$(\Psi_\omega^g)^{\mathcal{H}} = H_{11} \ominus H_{12} \odot H_{22}^{-1} \odot H_{21},$$

with $H_{11} := \tilde{H}|_{I(\partial\omega)\times I(\partial\omega)}$, $H_{12} := \tilde{H}|_{I(\partial\omega)\times I(\gamma_{12})}$,
$H_{21} := \tilde{H}|_{I(\gamma_{12})\times I(\partial\omega)}$, $H_{22} := \tilde{H}|_{I(\gamma_{12})\times I(\gamma_{12})}$.

Lemma 6.3.2 *The cost of building the $\mathcal{H}$-matrix H (see (6.17)) from $\mathcal{H}$-matrices H_1 and H_2 (see (6.16)) in case (6.19) by Algorithm 6.3.2 is*

$$N \le Ck^2 n \log^2 n, \quad \text{where } n = |I(\partial\omega)|, \quad C \in \mathbb{R}_+.$$

Proof: We follow the scheme in Fig. 6.5. Let $I := I(\partial\omega)$, $H' := H_1|^{I\times I}$ and $H'' := H_2|^{I\times I}$, where H' and H'' have the same block cluster structure as H. We convert H_1 to the matrix H' and H_2 to the matrix H''. The block structure of H is important and is shown in Fig. 6.5. As soon as the matrices H' and H'' are ready, we consider the construction of the matrix H as the addition $H' \oplus H''$ with further elimination of the unknowns with indices $i \in I(\gamma_{12})$. Using the inequality $n \le 2 \cdot |I(\partial\omega_1)|$ and Lemma 5.10.8 we obtain that the complexity of this addition is $\mathcal{O}(k^2 n \log^2 n)$. By Lemma 6.2.2 the elimination costs $\mathcal{O}(k^2 n \log n)$. The term $\mathcal{O}(k^2 n \log^2 n)$ dominates.

■

Example 6.3.1 *Figure 6.6 shows an example of building $(\Psi^g_\omega)^{\mathcal{H}} \in \mathbb{R}^{512\times 512}$ from $(\Psi^g_{\omega_1})^{\mathcal{H}} \in \mathbb{R}^{384\times 384}$ and $(\Psi^g_{\omega_2})^{\mathcal{H}} \in \mathbb{R}^{384\times 384}$. Let $I := I(\partial\omega \cup \gamma)$. The construction is performed in three steps: 1) build $H' := (\Psi^g_{\omega_1})^{\mathcal{H}}|^{I\times I}$ and $H'' := (\Psi^g_{\omega_2})^{\mathcal{H}}|^{I\times I}$,*
2) compute $\tilde{H} = H' \oplus H''$,
3) compute the Schur complement as in Statement 6.3.3.
Note that H', H'', $\tilde{H}$ have the same block structures. The symmetries of $(\Psi^g_{\omega_1})^{\mathcal{H}}$, $(\Psi^g_{\omega_2})^{\mathcal{H}}$ and $(\Psi^g_\omega)^{\mathcal{H}}$ are used.

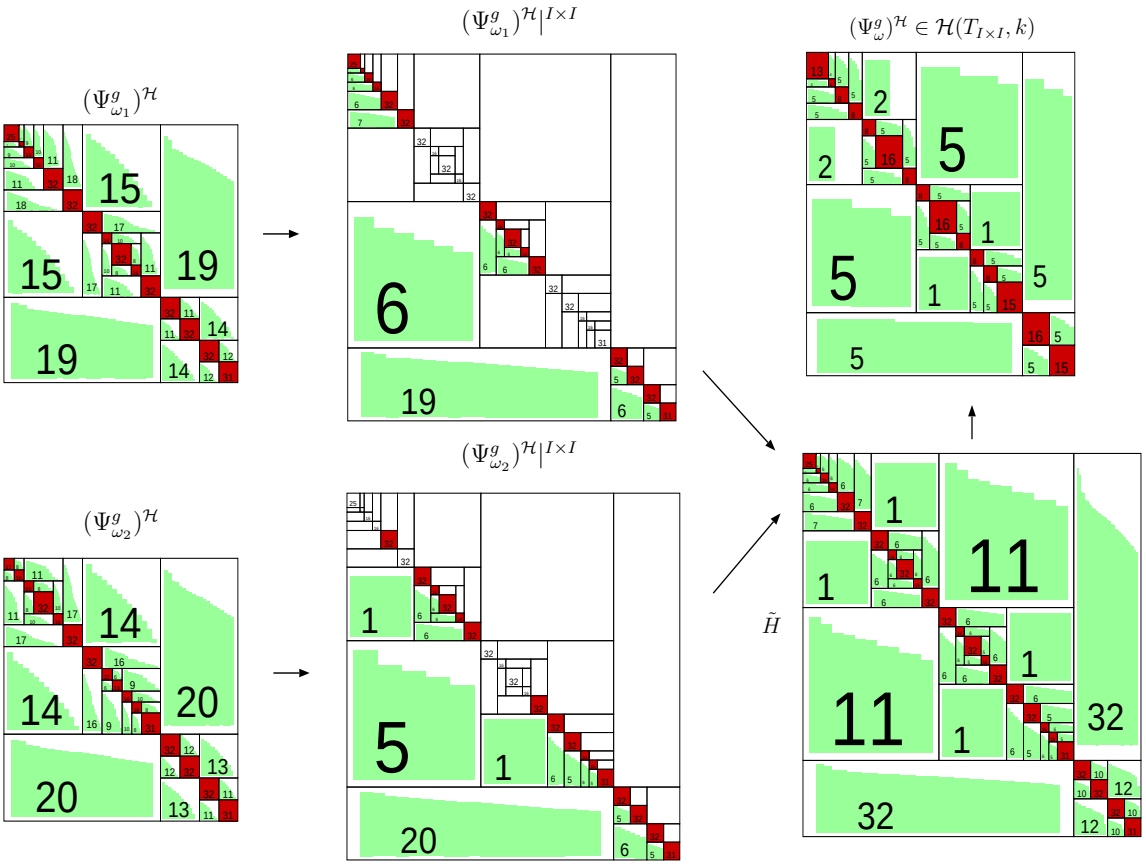


Figure 6.6: Building $(\Psi^g_\omega)^{\mathcal{H}} \in \mathbb{R}^{512\times 512}$ from $(\Psi^g_{\omega_1})^{\mathcal{H}}$ and $(\Psi^g_{\omega_2})^{\mathcal{H}}$ from $\mathbb{R}^{384\times 384}$. $\tilde{H} \in \mathbb{R}^{639\times 639}$ is an auxiliary matrix. The maximal size of the diagonal blocks is 32×32. The grey blocks indicate low-rank matrices. The steps inside the grey blocks show an exponential decay of the corresponding singular values. The white blocks indicate zero blocks. For the acceleration of building the symmetry of Ψ^g_ω is used.

6.3.2 Building $(\Psi^f_\omega)^{\mathcal{H}}$ from $(\Psi^f_{\omega_1})^{\mathcal{H}}$ and $(\Psi^f_{\omega_2})^{\mathcal{H}}$

Denote

$$H_1 := (\Psi^f_{\omega_1})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_1)\times I(\omega_1)}, k), \quad H_2 := (\Psi^f_{\omega_2})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_2)\times I(\omega_2)}, k) \tag{6.20}$$

and

$$\tilde{H} \in \mathcal{H}(T_{I(\partial\omega\cup\gamma)\times I(\omega)}, k), \quad H := (\Psi^f_\omega)^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega)\times I(\omega)}, k). \tag{6.21}$$

Let $T := T_{I(\partial\omega\cup\gamma)\times I(\omega)}$. We want to construct the matrix H from H_1 and H_2 (see Remark 4.3.3). Note that $\partial\omega \cup \gamma = \partial\omega_1 \cup \partial\omega_2$, $I(\partial\omega_i) = I(\Gamma_i) \cup I(\gamma)$, $\Gamma_1 \cup \Gamma_2 = \partial\omega$. To build the matrix H we need two cluster trees $T_{I(\partial\omega\cup\gamma)}$ and $T_{I(\omega)}$. The first cluster tree was already built for $(\Psi^g_\omega)^{\mathcal{H}}$. There are many possibilities of how to build $T_{I(\omega)}$, but we want a tree which makes a further elimination of unknowns x_i, $i \in I(\gamma)$ easier, i.e., one of the sons of the block cluster tree T has to coincide with the block $I(\gamma) \times I(\gamma)$. Therefore we choose the following decomposition:

$$I(\omega) = I(\omega_1 \setminus \gamma) \cup I(\omega_2 \setminus \gamma) \cup I(\gamma).$$

There are two cases:

$$I(\Gamma_i) \times I(\omega_i \setminus \gamma) \in T_{I(\partial\omega_i)\times I(\omega_i)} \text{ and } I(\gamma) \times I(\gamma) \in T_{I(\partial\omega_i)\times I(\omega_i)}, i = 1, 2. \tag{6.22}$$

$$I(\Gamma_i) \times I(\omega_i \setminus \gamma) \notin T_{I(\partial\omega_i)\times I(\omega_i)} \text{ or } I(\gamma) \times I(\gamma) \notin T_{I(\partial\omega_i)\times I(\omega_i)}, i = 1, 2. \tag{6.23}$$

Building algorithm in case (6.22):
Let H_1 and H_2 be defined as in (6.20), H as in (6.21) and $A_{12}A_{22}^{-1}$ as in (6.1).

Algorithm 6.3.3 *Build $H := (\Psi^f_\omega)^{\mathcal{H}}$ from $H_1 := (\Psi^f_{\omega_1})^{\mathcal{H}}$ and $H_2 := (\Psi^f_{\omega_2})^{\mathcal{H}}$*
build_ Ψ^f *(H_1, H_2, $A_{12}A_{22}^{-1}$)*
begin
allocate memory for $\tilde{H}$, H;
$\tilde{H}|_{I(\gamma)\times I(\gamma)} := H_1|_{I(\gamma)\times I(\gamma)} \oplus H_2|_{I(\gamma)\times I(\gamma)}$;
$\tilde{H}|_{I(\Gamma_1)\times I(\omega_1\setminus\gamma)} := H_1|_{I(\Gamma_1)\times I(\omega_1\setminus\gamma)}$;
$\tilde{H}|_{I(\Gamma_2)\times I(\omega_2\setminus\gamma)} := H_2|_{I(\Gamma_2)\times I(\omega_2\setminus\gamma)}$;
$\tilde{H}|_{I(\Gamma_1)\times I(\omega_2\setminus\gamma)} := 0$;
$\tilde{H}|_{I(\Gamma_2)\times I(\omega_1\setminus\gamma)} := 0$;
/* *in Fig. 6.7 denoted by [cg]* */
$\tilde{H}|_{I(\gamma)\times I(\omega\setminus\gamma)} := H_1|_{I(\gamma)\times I(\omega_1\setminus\gamma)} \oplus H_2|_{I(\gamma)\times I(\omega_2\setminus\gamma)}$;
/* *in Fig. 6.7 denoted by $b + f$* */
$\tilde{H}|_{I(\Gamma_1)\cup I(\Gamma_2)\times I(\gamma)} := H_1|_{I(\Gamma_1)\times I(\gamma)} \oplus H_2|_{I(\Gamma_2)\times I(\gamma)}$; /**sum of two low-rank matrices**/
$\tilde{H}$:=*extract_rows*($\tilde{H}$, r_1, r_2, i_1, i_2);/* *The output is r_1, r_2* */
$\tilde{H}$:=*extract_columns*($\tilde{H}$, c_1, c_2, j_1, j_2);/* *The output is c_1, c_2* */
$\tilde{H}$:=*add_rows*($\tilde{H}$, r_1, r_2, i_3, i_4);
$\tilde{H}$:=*add_columns*($\tilde{H}$, c_1, j_3, c_2, j_4);
$\tilde{H}_1 := \tilde{H}|_{I(\partial\omega)\times I(\omega)}$);
$\tilde{H}_2 := \tilde{H}|_{I(\gamma)\times I(\omega)}$);
return $H := \tilde{H}_1 \ominus A_{12} \odot A_{22}^{-1} \odot \tilde{H}_2$;
end;

Again, $I(\Gamma_1) \cap I(\Gamma_2) = I(\{x, y\}) \neq \varnothing$. This is the reason why we, first, remove two columns and two rows from $\tilde{H}$ and then add them to other blocks of $\tilde{H}$. The indices i_1, i_2, j_1, j_2 are the positions of two rows and two columns which have to be extracted. The indices i_3, i_4 and j_3, j_4 indicate the positions of two rows and two columns to which the extracted rows and columns r_1, r_2, c_1 and c_2 have to be added.

Lemma 6.3.3 *The cost of building the $\mathcal{H}$-matrix H (see (6.21)) from the $\mathcal{H}$-matrices H_1 and H_2 (see (6.20)) in case (6.22) by Algorithm 6.3.3 is*

$$N \leq Ck^2 n \log n, \qquad \textit{where } n = |I(\omega)|, C \in \mathbb{R}_+.$$

Proof: We follow the scheme in Fig. 6.7. Let $n_0 = |I(\partial\omega)|$ and $n_i = |I(\omega_i)|$, $i = 1, 2$.

1. From Table 5.3 follows that the complexity of adding $H_1|_{I(\gamma)\times I(\gamma)}$ and $H_2|_{I(\gamma)\times I(\gamma)}$ (denoted by $d + h$ in Fig. 6.7) is $\mathcal{O}(k^2 n_\gamma \log n_\gamma)$.

2. From Table 5.1 follows that the complexity of adding two low-rank matrices $b := H_1|_{I(\Gamma_1)\times I(\gamma)}$ and $f := H_1|_{I(\Gamma_2)\times I(\gamma)}$ (see Fig. 6.7) is $\mathcal{O}(k^2 n_0)$.

3. Building the new $\mathcal{H}$-matrix $[cg] \in \mathcal{H}(T_{I(\gamma)\times I(\omega_1\setminus\gamma)\cup I(\omega_2\setminus\gamma)}, k)$ from two $\mathcal{H}$-matrices $c := H_1|_{I(\gamma)\times I(\omega_1\setminus\gamma)}$, $g := H_2|_{I(\gamma)\times I(\omega_2\setminus\gamma)}$ costs $\mathcal{O}(1)$.

4. The removal of two columns and two rows from H_1 (see dotted lines in Fig. 6.7) by Lemma 5.10.3 costs $\mathcal{O}(k \log n_1)$.

5. Adding two columns and two rows to matrix H by Lemma 5.10.4 costs $\mathcal{O}(k^2 n_2)$ units. After that the 0-blocks become rank-2 matrices (denoted by R_2).

6. Adding four elements to the matrix $H|_{I(\Gamma_2)\times I(\omega_2\setminus\gamma)}$ in the general case by Lemma 5.10.4 costs $\mathcal{O}(k^2 n_2)$. In our case these elements are added to the dense matrices and the cost is $\mathcal{O}(1)$.

7. The cost of computing $F_1 - A_{12}A_{22}^{-1}F_2$ by Lemma 6.2.2 is $\mathcal{O}(k^2 n \log n)$.

Since the complexity $\mathcal{O}(k^2 n \log n)$ dominates, the total complexity is $\mathcal{O}(k^2 n \log n)$.

■

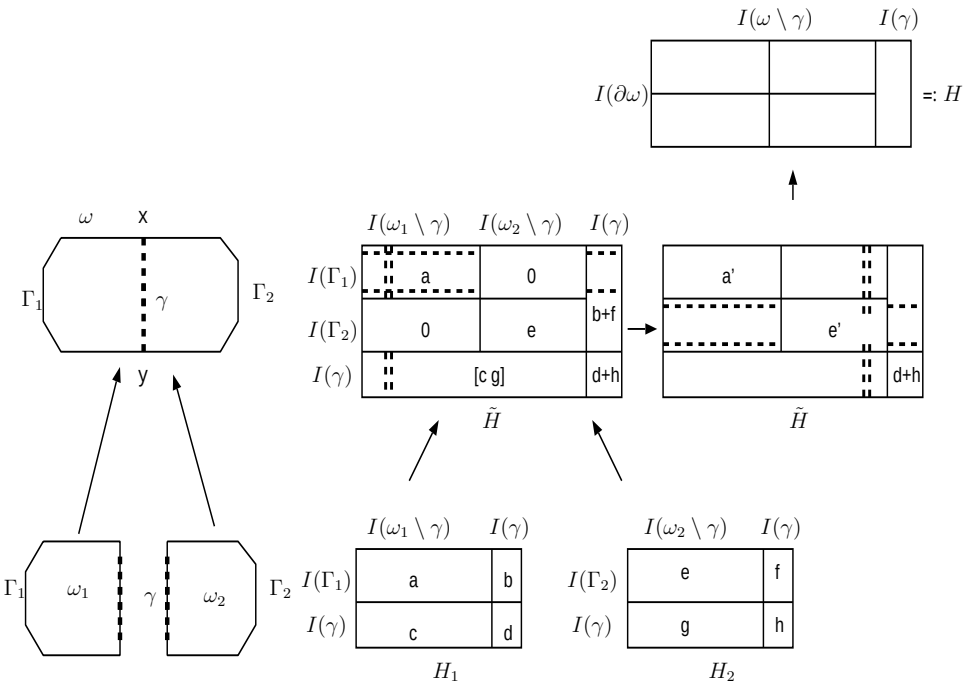


Figure 6.7: Building $(\Psi_\omega^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega)\times I(\omega)}$ from $(\Psi_{\omega_1}^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_1)\times I(\omega_1)}$ and $(\Psi_{\omega_2}^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_2)\times I(\omega_2)}$, $I(\Gamma_i) \cup I(\gamma) = I(\partial\omega_i)$, $i = 1, 2$, $I(\Gamma_1) \cap I(\Gamma_2) = I(\{x, y\})$. The small letters show the appearance of blocks in different matrices. The dotted lines in $\tilde{H}$ correspond to 2 rows and 2 columns, which were removed from $\tilde{H}$ and then added to other positions.

Remark 6.3.4 *Figure 6.7 illustrates Algorithm 6.3.3. First, it shows the construction of $\tilde{H}$, and then, according to (6.5),*

$$(\Psi_\omega^f)^{\mathcal{H}} = F_1 \ominus A_{12} \odot A_{22}^{-1} \odot F_2$$

with A_{12}, A_{22} from (6.1) and $F_1 := \tilde{H}|_{I(\partial\omega)\times I(\omega)}$ and $F_2 := \tilde{H}|_{I(\gamma_{12})\times I(\omega)}$.

Building algorithm in case (6.23):
Let $I := I(\partial\omega)$, $J := J(\omega)$ be two index sets and H_1 and H_2 be defined as in (6.20), H as in (6.21) and $A_{12}A_{22}^{-1}$ as in (6.1).

Algorithm 6.3.4 *Build* $H := (\Psi_\omega^f)^{\mathcal{H}}$ *from* $H_1 := (\Psi_{\omega_1}^f)^{\mathcal{H}}$ *and* $H_2 := (\Psi_{\omega_2}^f)^{\mathcal{H}}$
build_*Ψ^f*(H_1, H_2, $A_{12}A_{22}^{-1}$)
begin
allocate memory for $\tilde{H}$, H;
H':=*copy_block_structure(*H*);*
H'':=*copy_block_structure(*H*);*
h2h(H_1, H',...*);* /∗*convert* H_1 *to* H' *by Algorithm 5.9.2*∗/
h2h(H_2, H'',...*);*
$\tilde{H} := H'' \oplus H'$;
$\tilde{H}_1 := \tilde{H}|_{I(\partial\omega)\times I(\omega)}$);
$\tilde{H}_2 := \tilde{H}|_{I(\gamma_{12})\times I(\omega)}$);
return $H := \tilde{H}_1 \ominus A_{12} \odot A_{22}^{-1} \odot \tilde{H}_2$;
end;

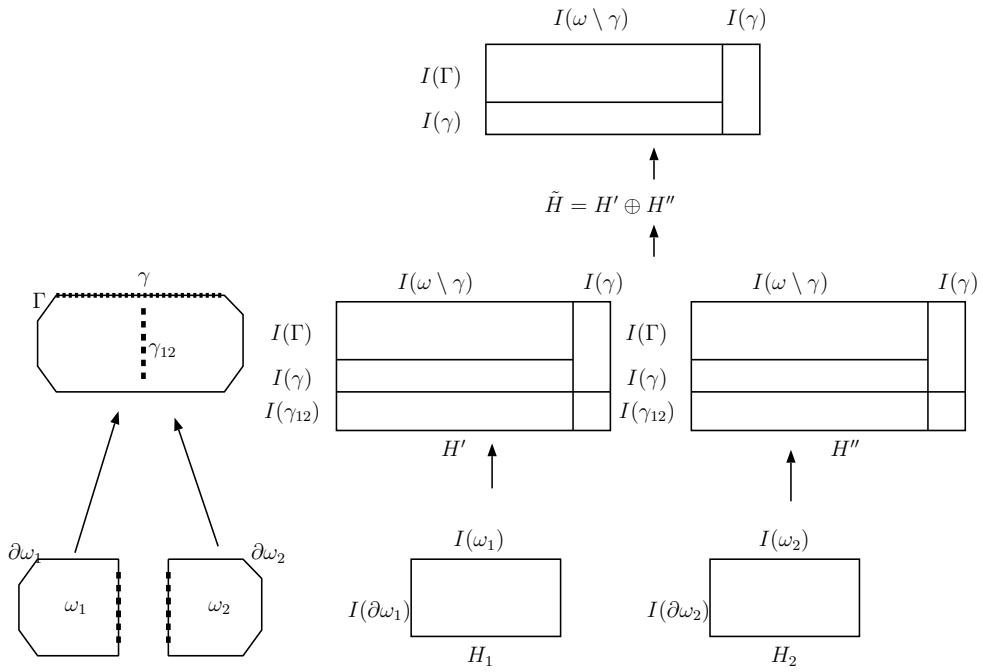


Figure 6.8: Building $H := (\Psi_\omega^f)^{\mathcal{H}} \in \mathbb{R}^{I\times J}$, $I := I(\Gamma) \cup I(\gamma)$, $J := I(\omega \setminus \gamma) \cup I(\gamma)$, from $H_1 := (\Psi_{\omega_1}^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_1)\times I(\omega_1)}$ and $H_2 := (\Psi_{\omega_2}^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_2)\times I(\omega_2)}$.

Remark 6.3.5 *Figure 6.8 illustrates Algorithm 6.3.4. First, it shows the construction of* $\tilde{H}$. $\tilde{H} = H' \oplus H''$ *with*

$$H' := H_1|^{I(\partial\omega\cup\gamma_{12})\times I(\partial\omega\cup\gamma_{12})}, \quad H'' := H_2|^{I(\partial\omega\cup\gamma_{12})\times I(\partial\omega\cup\gamma_{12})} \text{ and } \gamma_{12} := \partial\omega_1 \setminus \partial\omega.$$

Then, according to (6.5),

$$(\Psi_\omega^f)^{\mathcal{H}} = F_1 \ominus A_{12} \odot A_{22}^{-1} \odot F_2,$$

where A_{12}, A_{22} are defined in (6.1) and $F_1 := \tilde{H}|_{I(\partial\omega)\times I(\omega)}$ and $F_2 := \tilde{H}|_{I(\gamma_{12})\times I(\omega)}$.

Lemma 6.3.4 *The cost of building the $\mathcal{H}$-matrix H (see (6.21)) from the $\mathcal{H}$-matrices H_1 and H_2 (see (6.20)) in case (6.23) using Algorithm 6.3.4 is*

$$N \le Ck^2 n \log^2 n, \quad \text{where } n = |I(\omega)|, \quad C \in \mathbb{R}_+.$$

Proof: We follow the scheme in Fig. 6.8. Let $I := I(\partial\omega)$, $J := J(\omega)$, $n := |I(\omega)|$. Building of H is equivalent to the addition $H' \oplus H''$, where $H' := H_1|^{I\times J}$ and $H'' := H_2|^{I\times J}$ and the further elimination of the unknowns x_i, $i \in I(\gamma_{12})$. It follows from Lemma 5.10.8 that adding H' and H'' costs $\mathcal{O}(k^2 n \log^2 n)$. It follows from Lemma 6.2.2 that the cost of the elimination is $\mathcal{O}(k^2 n \log n)$.

■

Building $(\Psi^f_\omega)^{\mathcal{H}}$ from $(\Psi^f_{\omega_1})^{\mathcal{H}}$ and $(\Psi^f_{\omega_2})^{\mathcal{H}}$ for the two-grid modification

The index $_h$ indicates the quantities of the fine grid and the index $_H$ of the coarse grid. Denote

$$H_1 := (\Psi^f_{\omega_1})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_{1,h})\times I(\omega_{1,H})}, k), \tag{6.24}$$

$$H_2 := (\Psi^f_{\omega_2})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_{2,h})\times I(\omega_{2,H})}, k). \tag{6.25}$$

We want to construct the matrix

$$H := (\Psi^f_{\omega})^{\mathcal{H}} \in \mathcal{H}(T_{I(\partial\omega_h)\times I(\omega_H)}, k). \tag{6.26}$$

Note that $I(\partial\omega_h) \cup I(\gamma_h) = I(\partial\omega_{1,h}) \cup I(\partial\omega_{2,h})$, $I(\partial\omega_{i,h}) = I(\Gamma_{i,h}) \cup I(\gamma_h)$. We construct the tree $T_{I(\omega_H)}$ so that the further elimination of the unknowns $\mathbf{u}_i$, $i \in I(\gamma_H)$ becomes easier, i.e., we want that $I(\gamma_h) \times I(\gamma_H) \in T_{I(\partial\omega_h)\times I(\omega_H)}$. We choose the following decomposition

$$I(\omega) = I(\omega_{1,H} \setminus \gamma_H) \cup I(\omega_{2,H} \setminus \gamma_H) \cup I(\gamma_H),$$

$$I(\partial\omega_h) = I(\Gamma_{1,h}) \cup I(\Gamma_{2,h}).$$

There are two cases:

$$I(\Gamma_{i,h}) \times I(\omega_{i,H}\backslash\omega_H), I(\gamma_h) \times I(\gamma_H) \in T_{I(\partial\omega_{i,h})\times I(\omega_H)}, i = 1, 2, \tag{6.27}$$

$$I(\Gamma_{i,h}) \times I(\omega_{i,H}\backslash\omega_H) \notin T_{I(\partial\omega_{i,h})\times I(\omega_H)} \text{ or } I(\gamma_h) \times I(\gamma_H) \notin T_{I(\partial\omega_{i,h})\times I(\omega_H)}, i = 1, 2. \tag{6.28}$$

Algorithms 6.3.3, 6.3.4 with small modifications are used for cases (6.27) and (6.28) accordingly.

Lemma 6.3.5 *Let matrix H be as in (6.26) and H_1 and H_2 as in (6.24), (6.25), then the cost of building H from H_1 and H_2 is*

$$N = \mathcal{O}(k^2 n \log n) \quad \text{in case (6.27)}$$
$$N = \mathcal{O}(k^2 n \log^2 n) \quad \text{in case (6.28)},$$

where $n = \max\{|I(\partial\omega_h)|, |I(\omega_H)|\}$.

Proof: Analogous to the proofs of Lemmas 6.3.3, 6.3.4.

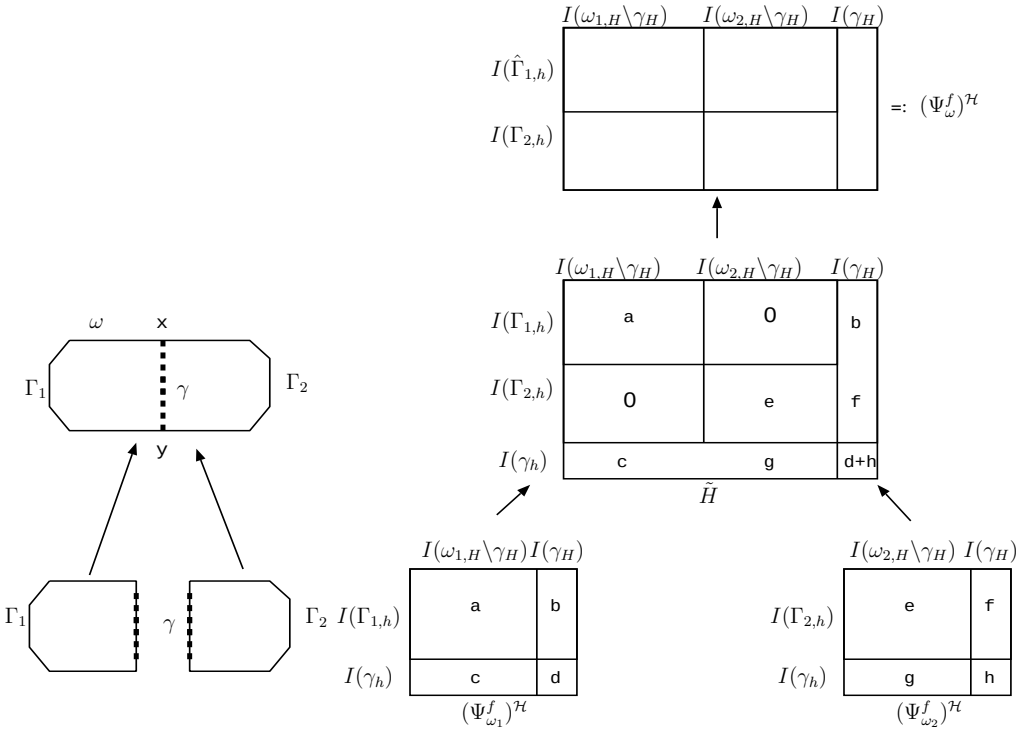


Figure 6.9: Building $(\Psi_\omega^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_h)\times I(\omega_H)}$ from $(\Psi_{\omega_1}^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_{1,h})\times I(\omega_{1,H})}$ and $(\Psi_{\omega_2}^f)^{\mathcal{H}} \in \mathbb{R}^{I(\partial\omega_{2,h})\times I(\omega_{2,H})}$ for two grids with step sizes H and h. $I(\hat{\Gamma}_{1,h}) = I(\Gamma_{1,h}) \setminus I(\{x,y\})$, $I(\partial\omega_h) = I(\hat{\Gamma}_{1,h}) \cup I(\Gamma_{2,h})$, $I(\omega_H) = I(\omega_{1,H} \setminus \gamma_H) \cup I(\omega_{2,H} \setminus \gamma_H) \cup I(\gamma_H)$. The small letters show the appearance of blocks in different matrices.

Remark 6.3.6 *Figure 6.9 illustrates Algorithm 6.3.3, but for the two-grid modification of HDD. First, it shows the construction of $\tilde{H}$, and then, according to (6.5),*

$$(\Psi_\omega^f)^{\mathcal{H}} = F_1 \ominus A_{12} \odot A_{22}^{-1} \odot F_2$$

with A_{12}, A_{22} from (6.1) and $F_1 := \tilde{H}|_{I(\partial\omega_h)\times I(\omega_H)}$ and $F_2 := \tilde{H}|_{I(\gamma_h)\times I(\omega_H)}$.

7 Complexity and Storage Requirement of HDD

Let us recall four types of mappings which are present in the HDD method: domain-to-boundary Ψ^f_ω, boundary-to-boundary Ψ^g_ω, domain-to-interface Φ^f_ω and boundary-to-interface Φ^g_ω, for all domains $\omega \in T_{\mathcal{T}_h}$. In this Chapter we estimate the computational complexities of HDD and its modifications. We estimate also the storage requirements of Φ^f_ω and Φ^g_ω. Finally, in the conclusion, we consider a special case, namely, when the right-hand side is equal to 0.

7.1 Notation and Auxiliary Lemmas

Let $x \in D \subset \mathbb{R}^n$, $y \in \mathbb{R}^m$. Let $\psi : x \mapsto y$ be a given mapping. An algorithm for computing $\psi(x)$ is a sequence of elementary operations. The computational complexity of the algorithm is characterised by the number of elementary operations N_ψ. We take into account only the addition and the multiplication of real numbers, but in some special cases (e.g., the removal of a column from an $\mathcal{H}$-matrix) we also take the cost of the coping into account.

Definition 7.1.1 *If the computational complexity depends linearly on the data size n, the algorithm has a linear computational complexity.*

Definition 7.1.2 *If the storage requirement depends linearly on the data size n, the algorithm has a linear storage complexity.*

Definition 7.1.3 *We call the complexity $\mathcal{O}(n \log^q n)$, where $q = 1, 2, 3$, an almost linear complexity.*

Remark 7.1.1 *For large input vectors (e.g., $n = 1000000$ is very common nowadays) the time for an algorithm with linear complexity can be one hour and for an algorithm with quadratic complexity one month. This is the reason why it is important to develop and to use the algorithms with a linear complexity.*

There is an empirical observation that says that there is a linear dependence between the computer memory and the processor frequency. To get the optimal ratio (productivity) / (computational resources) the complexity of the algorithm must be a linear function of the computer memory and the processor frequency. This is another reason to develop and to use algorithms with linear complexity.

In this section we show that the HDD method (Case (a)) and its three modifications (Cases (b),(c),(d)) have almost linear complexity. We characterise each case

by the domain of definition and range of Ψ_ω and Φ_ω.

Case (a) - The standard HDD method
$\Psi^f_\omega : \mathbb{R}^{I(\omega_h)} \to \mathbb{R}^{I(\partial\omega_h)}$, $\omega \in T_{\mathcal{T}_h}$,
$\Psi^g_\omega : \mathbb{R}^{I(\partial\omega_h)} \to \mathbb{R}^{I(\partial\omega_h)}$, $\omega \in T_{\mathcal{T}_h}$,
$\Phi^f_\omega : \mathbb{R}^{I(\omega_h)} \to \mathbb{R}^{I(\gamma_h)}$, $\omega \in T_{\mathcal{T}_h}$,
$\Phi^g_\omega : \mathbb{R}^{I(\partial\omega_h)} \to \mathbb{R}^{I(\gamma_h)}$, $\omega \in T_{\mathcal{T}_h}$.

Case (b) - HDD with truncation of the small scales
$\Psi^f_\omega : \mathbb{R}^{I(\omega_h)} \to \mathbb{R}^{I(\partial\omega_h)}$, $\omega \in T_{\mathcal{T}_h}$,
$\Psi^g_\omega : \mathbb{R}^{I(\partial\omega_h)} \to \mathbb{R}^{I(\partial\omega_h)}$, $\omega \in T_{\mathcal{T}_h}$,
$\Phi^f_\omega : \mathbb{R}^{I(\omega_h)} \to \mathbb{R}^{I(\gamma_h)}$, $\omega \in T^{\geq H}_{\mathcal{T}_h}$,
$\Phi^g_\omega : \mathbb{R}^{I(\partial\omega_h)} \to \mathbb{R}^{I(\gamma_h)}$, $\omega \in T^{\geq H}_{\mathcal{T}_h}$.

Let $P : \mathbb{R}^{I(\omega_H)} \to \mathbb{R}^{I(\omega_h)}$ be a prolongation matrix as it is defined in Section 4.3.6.

Case (c) - HDD on two grids
$\Psi^f_\omega := \tilde{\Psi}^f_\omega P_{h\leftarrow H}$, where $\tilde{\Psi}^f_\omega : \mathbb{R}^{I(\omega_h)} \to \mathbb{R}^{I(\partial\omega_h)}$, $\Psi^f_\omega : \mathbb{R}^{I(\omega_H)} \to \mathbb{R}^{I(\partial\omega_h)}$, $\omega \in T_{\mathcal{T}_h}$,
$\Psi^g_\omega : \mathbb{R}^{I(\partial\omega_h)} \to \mathbb{R}^{I(\partial\omega_h)}$, $\omega \in T_{\mathcal{T}_h}$,
$\Phi^f_\omega := \tilde{\Phi}^f_\omega P_{h\leftarrow H}$, where $\tilde{\Phi}^f_\omega : \mathbb{R}^{I(\omega_h)} \to \mathbb{R}^{I(\gamma_h)}$, $\Phi^f_\omega : \mathbb{R}^{I(\omega_H)} \to \mathbb{R}^{I(\gamma_h)}$, $\omega \in T_{\mathcal{T}_h}$,
$\Phi^g_\omega : \mathbb{R}^{I(\partial\omega_h)} \to \mathbb{R}^{I(\gamma_h)}$, $\omega \in T_{\mathcal{T}_h}$.

Case (d) - HDD on two grids and with truncation of the small scales
$\Psi^f_\omega := \tilde{\Psi}^f_\omega P_{h\leftarrow H}$, where $\tilde{\Psi}^f_\omega : \mathbb{R}^{I(\omega_h)} \to \mathbb{R}^{I(\partial\omega_h)}$, $\Psi^f_\omega : \mathbb{R}^{I(\omega_H)} \to \mathbb{R}^{I(\partial\omega_h)}$, $\omega \in T_{\mathcal{T}_h}$,
$\Psi^g_\omega : \mathbb{R}^{I(\partial\omega_h)} \to \mathbb{R}^{I(\partial\omega_h)}$, $\omega \in T_{\mathcal{T}_h}$,
$\Phi^f_\omega := \tilde{\Phi}^f_\omega P_{h\leftarrow H}$, where $\tilde{\Phi}^f_\omega : \mathbb{R}^{I(\omega_h)} \to \mathbb{R}^{I(\gamma_h)}$, $\Phi^f_\omega : \mathbb{R}^{I(\omega_H)} \to \mathbb{R}^{I(\gamma_h)}$, $\omega \in T^{\geq H}_{\mathcal{T}_h}$,
$\Phi^g_\omega : \mathbb{R}^{I(\partial\omega_h)} \to \mathbb{R}^{I(\gamma_h)}$, $\omega \in T^{\geq H}_{\mathcal{T}_h}$.

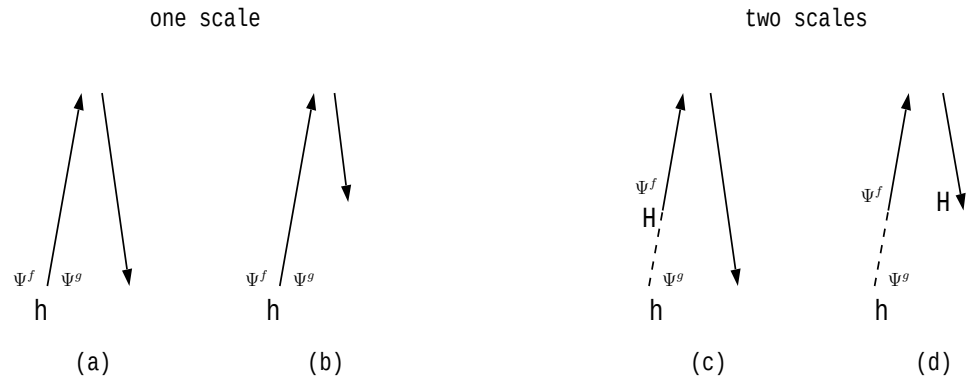


Figure 7.1: Four cases of the HDD method: (a) the standard HDD method with a fine scale h, (b) HDD with a fine scale h and with truncation of the small scales, (c) HDD on two grids, (d) HDD on two grids and with truncation of the small scales.

We consider four modifications of the HDD method (see Figure 7.1). Case (a) is HDD without any modifications, i.e., there is only one scale h and no truncation.

In Case (b) there is one scale, but the algorithm "Root to Leaves" stops at a coarse scale H. Case (c) is HDD on two grids and a right-hand side f given at a coarse scale H. The fourth Case (d) is the same as the third case, but the algorithm "Root to Leaves" truncates the scales smaller than H. Case (d) is the cheapest one.
The long arrows show the algorithms "Leaves to Root" and "Root to Leaves", the short arrow in (c) indicates that the algorithm starts not with the smallest scale h, but with a scale H. The short arrow in (d) indicates that there is truncation of the small scales. The dotted lines mean that in the corresponding parts of the domain decomposition tree $T_{\mathcal{T}_h}$ the right-hand side is given on the coarse space $V_H \subset V_h$ and the prolongation matrix is used.

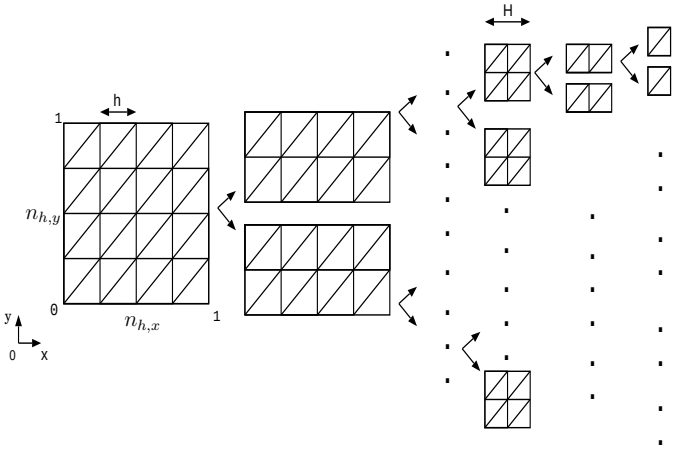


Figure 7.2: Model domain $\Omega = (0,1)^2$ with rectangular grid and its hierarchical decomposition. $n_{h,x}$ and $n_{h,y}$ are the numbers of grid points in 0x and 0y directions.

Remark 7.1.2 *(Model Domain)*
To keep further theoretical calculations simple, we consider the model domain $\Omega = (0,1)^2$ *with the rectangular grid and its hierarchical decomposition as in Fig. 7.2.*

Consider the model problem as in Remark 7.1.2.
Let $n_{h,x}(\omega)$ and $n_{h,y}(\omega)$ be the numbers of grid points in $\omega \in T_{\mathcal{T}_h}$ in the 0x and 0y directions.
The number of grid points in the domain $\omega \in T_{\mathcal{T}_h}$ is $n_h(\omega) = n_{h,x}(\omega) \cdot n_{h,y}(\omega)$.
A domain ω on the level i of $T_{\mathcal{T}_h}$ we denote by ω_i. The root of $T_{\mathcal{T}_h}$ has level 0.
Suppose that $n_{h,x}(\Omega) = n_{h,y}(\Omega) = 2^p + 1$ and $n_{H,x}(\Omega) = n_{H,y}(\Omega) = 2^q + 1$. Then the number of nodal points in Ω is $n_h := n_h(\Omega) = (2^p + 1)^2$ and if the grid step size is H, then $n_H := n_H(\Omega) = (2^q + 1)^2$.
The number of nodal points on the interface (see Fig. 7.4) is denoted by $n_{h,H}$ and $n_{h,H} \leq 2n_{h,x}n_{H,x} = 2\sqrt{n_h n_H}$.
Suppose that $|I(\partial\omega_i)| \leq 6n_{h,x}(\omega_i)$ and
$n_{\gamma_{\omega_i}} := |I(\gamma_{\omega_i})| = n_{h,x}(\omega_i) - 2$.
The number of subdomains on each level i of $T_{\mathcal{T}_h}$ is 2^l, $i = 0, ..., 2p$.

The depth of $T_{\mathcal{T}_h}$ is not larger than $\log_2 n_h(\Omega) + 2$.
Note that $2p \leq \log_2 n_h(\Omega) \leq 2p + 1$, $p \geq 2$.
The depth of the domain decomposition tree with scale H (denoted by $T_{\mathcal{T}_h}^{\geq H}$) is $2q \leq depth(T_{\mathcal{T}_h}^{\geq H}) := \log_2 n_H(\Omega) \leq 2q + 1$.

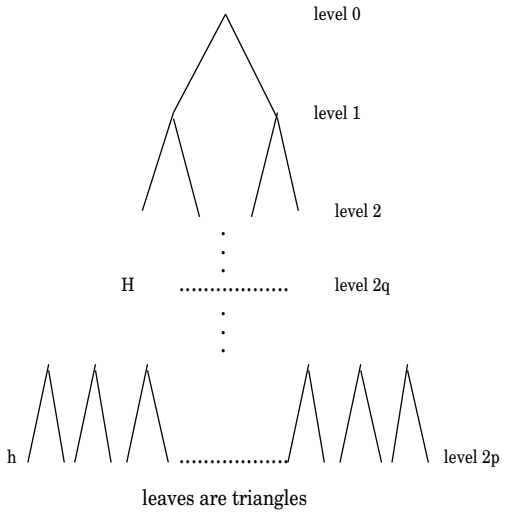


Figure 7.3: Two levels $2q$ and $2p$ of the domain decomposition tree $T_{\mathcal{T}_h}$.

Definition 7.1.4 *The storage for all mappings Φ_ω, $\omega \in T_{\mathcal{T}_h}$, is denoted by*

$$S(\Phi) := \sum_{\omega \in T_{\mathcal{T}_h}} S(\Phi_\omega).$$

Definition 7.1.5 *The number of arithmetic operations for computing all mappings Ψ_ω, $\omega \in T_{\mathcal{T}_h}$, is denoted by*

$$N(\Psi) := \sum_{\omega \in T_{\mathcal{T}_h}} N(\Psi_\omega).$$

For further estimates we will need the following equalities (proofs of these facts see in Appendix):

$$\sum_{i=0}^{p} i^2 = \frac{1}{3}p^3 + \mathcal{O}(p^2), \tag{7.1}$$

$$\sum_{i=0}^{p} i2^i = (p-1)2^{p+1} + 2, \tag{7.2}$$

$$\sum_{i=0}^{p} (p-i)2^i = 2^{p+1} - p - 2, \tag{7.3}$$

$$\sum_{i=0}^{p} (p-i)^2 2^i \leq 3 \cdot 2^{p+1}, \tag{7.4}$$

$$\sum_{i=0}^{p} i^2 2^i = (p^2 - 2p + 3)2^{p+1} - 6. \tag{7.5}$$

Remark 7.1.3 *In the following we will assume that*

$$|I(\partial\omega_i)| \approx 2|I(\partial\omega_{i+2})| \quad \textit{and}$$

$$|I(\omega_i)| \approx 2|I(\omega_{i+1})|.$$

7.2 Complexity of the Recursive Algorithm "Leaves to Root"

Now we would like to compute the computational cost of the algorithm "Leaves to Root". The Algorithm 7.2.1 computes recursively the mappings Φ^g_ω and Φ^f_ω for all $\omega \in T_{\mathcal{T}_h}$. The domain decomposition tree $T_{\mathcal{T}_h}(\Omega)$ is an input parameter.

Algorithm 7.2.1 *(Leaves to Root)*
leaves_to_root(structure $T_{\mathcal{T}_h}(\omega)$)
begin
T_1 :=*get_left_son*($T_{\mathcal{T}_h}(\omega)$);
T_2 :=*get_right_son*($T_{\mathcal{T}_h}(\omega)$);
if $(T_1 != \varnothing)$ ***and*** $(T_2 != \varnothing)$ ***then***
if (T_1 *is not computed*) ***then***
leaves_to_root(T_1);
if (T_2 *is not computed*) ***then***
leaves_to_root(T_2);
Ψ^g_ω := *build_*Ψ^g($\Psi^g_{\omega_1}, \Psi^g_{\omega_2}$); /* *See Algorithms 6.3.1, 6.3.2* */
Ψ^f_ω := *build_*Ψ^f($\Psi^f_{\omega_1}, \Psi^f_{\omega_2}$); /* *See Algorithms 6.3.3, 6.3.4* */
Φ^g_ω := *compute_*Φ^g(..);
Φ^f_ω := *compute_*Φ^f(..);
delete(Ψ_{ω_1});
delete(Ψ_{ω_2});
end if;
end;

Lemma 7.2.1 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the computational complexity of all mappings $\Psi^g_\omega : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\partial\omega)}$, $\omega \in T_{\mathcal{T}_h}$, has the following upper bound:*

$$N(\Psi^g) \leq Ck^2 n_h, \quad C \in \mathbb{R}_+.$$

Proof: Let $N(\Psi^g_\omega)$ be the cost of building Ψ^g_ω from $\Psi^g_{\omega_1}$ and $\Psi^g_{\omega_2}$, where $\omega, \omega_1, \omega_2 \in T_{\mathcal{T}_h}$ and ω_1, ω_2 are sons of ω. Note that all ω at a fixed level of $T_{\mathcal{T}_h}$ have the same

number of degrees of freedom. The total complexity is

$$\begin{aligned}
N(\Psi^g) &:= \sum_{\omega\in T_{\mathcal{T}_h}} N(\Psi^g_\omega)\\
&= N(\Psi^g_{\omega_0}) + 2N(\Psi^g_{\omega_1}) + 4N(\Psi^g_{\omega_2}) + 8N(\Psi^g_{\omega_3}) + ... + 2^{2p+1}N(\Psi^g_{\omega_{2p+1}})\\
&\le 3N(\Psi^g_{\omega_0}) + 12N(\Psi^g_{\omega_2}) + ... + (2^{2p} + 2^{2p+1})N(\Psi^g_{\omega_{2p}})\\
&\le \sum_{i=0}^{p} (2^{2i+1} + 2^{2i})N(\Psi^g_{\omega_{2i}})\\
&\overset{\text{Lem.6.2.1}}{=} \sum_{i=0}^{p} (2^{2i+1} + 2^{2i})(C_{2i}k^2 n_{h,x}(\omega_{2i}) \log^2 n_{h,x}(\omega_{2i}))\\
&\le 3 \max_i C_{2i}k^2 \sum_{i=0}^{p} 2^i \cdot 2^i \frac{n_{h,x}}{2^i} \log^2 \frac{n_{h,x}}{2^i}\\
&\overset{C':=\max_i C_{2i}}{=} 3C'k^2 n_{h,x} \sum_{i=0}^{p} 2^i \cdot (\log n_{h,x} - i)^2\\
&\le 3C'k^2 n_{h,x} \sum_{i=0}^{p} 2^i (p+1-i)^2 = 6C'k^2 n_{h,x} \sum_{i=-1}^{p-1} 2^i (p-i)^2\\
&\overset{(7.4)}{\le} Ck^2 n_{h,x} \cdot 2^p \le Ck^2 n_h, \quad C \in \mathbb{R}_+.
\end{aligned}$$

■

Lemma 7.2.2 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the computational complexity of all mappings $\Psi^f_\omega : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\partial\omega)}$, $\omega \in T_{\mathcal{T}_h}$, has the following upper bound:*

$$N(\Psi^f) \le Ck^2 n_h \log^3 n_h, \quad C \in \mathbb{R}_+.$$

Proof: Let $N(\Psi^f_\omega)$ be the complexity of building Ψ^f_ω from $\Psi^f_{\omega_1}$ and $\Psi^f_{\omega_2}$, where $\omega, \omega_1, \omega_2 \in T_{\mathcal{T}_h}$ and ω_1, ω_2 are sons of ω. Lemma 6.3.4 yields $N(\Psi^f_{\omega_i}) \le C_i k^2 n_h(\omega_i) \log^2 n_h(\omega_i)$. Since all ω at a fixed level of $T_{\mathcal{T}_h}$ are equal, the following estimate holds

$$\begin{aligned}
N(\Psi^f) &:= \sum_{\omega\in T_{\mathcal{T}_h}} N(\Psi^f_\omega)\\
&= N(\Psi^f_{\omega_0}) + 2N(\Psi^f_{\omega_1}) + 4N(\Psi^f_{\omega_2}) + 8N(\Psi^f_{\omega_3}) + ... + 2^{2p}N(\Psi^f_{\omega_{2p}})\\
&\le \sum_{i=0}^{2p} 2^i (C_i k^2 n_h(\omega_i) \log^2 n_h(\omega_i))\\
&\le \sum_{i=0}^{2p} 2^i C'_i k^2 \frac{n_h}{2^i} \log^2 \frac{n_h}{2^i}\\
&\overset{C':=\max_i C'_i}{\le} C'k^2 n_h \sum_{i=0}^{2p} (2p+1-i)^2\\
&\overset{(7.1)}{\le} C'k^2 n_h (\frac{1}{3}(2p+1)^3) \le Ck^2 n_h \log^3 n_h, C \in \mathbb{R}_+.
\end{aligned}$$

Lemma 7.2.3 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the computational complexity of all mappings Ψ_ω, $\omega \in T_{\mathcal{T}_h}$, has the following upper bound:*

$$N(\Psi) = N(\Psi^f) + N(\Psi^g) \leq Ck^2 n_h \log^3 n_h, \quad C \in \mathbb{R}_+.$$

Proof: See Lemmas 7.2.1 and 7.2.2.

7.3 Complexity of the Recursive Algorithm "Root to Leaves"

The Algorithm 7.3.1 computes recursively the solution $u|_{I(\gamma_\omega)}$ for all $\omega \in T_{\mathcal{T}_h}$. The input data are the domain decomposition tree $T_{\mathcal{T}_h}(\Omega)$, the mappings Φ^g_ω and Φ^f_ω for all $\omega \in T_{\mathcal{T}_h}$, and the solution $u|_{I(\partial\omega)}$ for a current subdomain ω.

$$u(x) = \begin{cases} g(x) & \text{for all } x \in \partial\omega, \\ 0 & \text{otherwise.} \end{cases} \tag{7.6}$$

Note that in the Algorithm 7.3.1 g_ω and f_ω are the Dirichlet data and the right-hand side for the local problem, defined on $\omega \in T_{\mathcal{T}_h}$.

Algorithm 7.3.1 *(Root to Leaves)*
root_to_leaves$(T_{\mathcal{T}_h}(\omega), u)$
begin
T_1 :=*left_son*$(T_{\mathcal{T}_h}(\omega))$;
T_2 :=*right_son*$(T_{\mathcal{T}_h}(\omega))$;
$u_{\gamma_\omega} := \Phi^g_\omega \cdot g_\omega + \Phi^f_\omega \cdot f_\omega$;
for all $i \in I(\gamma_\omega)$ ***do***
u[*local_to_global*[i]]:=$u_{\gamma_\omega}[i]$;
if $(T_1 != \varnothing)$ ***then***
root_to_leaves(T_1, u);
if $(T_2 != \varnothing)$ ***then***
root_to_leaves(T_2, u);
end;

Here local_to_global[...] is an auxiliary index mapping, which for each global index in $I(\Omega)$ returns its local index in $I(\omega)$. Note that we computed the mappings Φ^g_ω and Φ^f_ω, $\omega \in T_{\mathcal{T}_h}$, during the computation of the auxiliary mappings Ψ^f_ω and Ψ^g_ω. Below we estimate the complexity of computing the solution u.

Notation 7.3.1 *We denote by N_g the complexity of all matrix-vector multiplications $\Phi^g_\omega \cdot g_\omega$, where $\Phi^g_\omega \in \mathbb{R}^{I(\gamma_\omega)\times I(\partial\omega)}$, $g \in \mathbb{R}^{I(\partial\omega)}$.*
We denote by N_f the complexity of all matrix-vector multiplications $\Phi^f_\omega \cdot f_\omega$, where $\Phi^f_\omega \in \mathbb{R}^{I(\gamma_\omega)\times I(\omega)}$, $f \in \mathbb{R}^{I(\omega)}$.

Lemma 7.3.1 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the computational complexity of all matrix-vector*

multiplications $\Phi^g_\omega \cdot g_\omega$, *where* $\Phi^g_\omega : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\gamma)}$, $\omega \in T_{\mathcal{T}_h}$, $g \in \mathbb{R}^{I(\partial\omega)}$, *has the following upper bound:*

$$N_g \leq 21kn_h.$$

Proof: We have $n_{h,\gamma} \leq n_{h,x}(\omega)$ for the model problem as in Remark 7.1.2. The matrix Φ^g_ω is approximated in the low-rank format and belongs to $\mathcal{R}(k, I(\gamma), I(\partial\omega))$. The complexity of the matrix-vector multiplication can be estimated as follows:

$$N_g(\omega) \leq 2k \cdot (|I(\gamma)| + |I(\partial\omega)|) \leq 2k(n_{h,x}(\omega) + 6n_{h,x}(\omega)) = 14kn_{h,x}(\omega).$$

The total complexity is

$$\begin{aligned} N_g &= \sum_{\omega \in T_{\mathcal{T}_h}} N_g(\omega) \\ &\leq \sum_{i=0}^{p-2} (2^{2i+1} + 2^{2i})(14kn_{h,x}(\omega_i)) \\ &\leq 42k \sum_{i=0}^{p-2} 2^{2i} \frac{n_{h,x}}{2^i} \\ &= 42kn_{h,x} \sum_{i=0}^{p-2} 2^i \leq 21kn_{h,x}2^p \leq 21kn_h. \end{aligned}$$

■

Lemma 7.3.2 *Let* k *be the rank which is used in the* $\mathcal{H}$*-matrix arithmetic. For the model domain from Remark 7.1.2 the computational complexity of all matrix-vector products* $\Phi^f_\omega \cdot f_\omega$, *where* $\Phi^f_\omega : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\gamma)}$, $\omega \in T_{\mathcal{T}_h}$, $f_\omega \in \mathbb{R}^{I(\omega)}$, *has the following upper bound:*

$$N_f \leq Ckn_h \log^2 n_h, \quad C \in \mathbb{R}_+.$$

Proof: Each level $i \in [0, 2p-3]$ of the tree $T_{\mathcal{T}_h}$ contains 2^i matrices and each matrix is multiplied by a vector f_ω. The complexity is

$$\begin{aligned} N_f &= \sum_{i=0}^{2p-3} 2^i N_f(\omega_i) \\ &= \sum_{i=0}^{2p-3} 2^i (C_i kn_h(\omega_i) \log n_h(\omega_i)) \\ &\leq k \cdot \sum_{i=0}^{2p-3} 2^i (C'_i \frac{n_h}{2^i} \log \frac{n_h}{2^i}) \\ &\overset{C' := \max_i C'_i}{\leq} C' n_h k \sum_{i=0}^{2p-3} (2p + 1 - i) \leq C' n_h kp(2p+4). \end{aligned}$$

We recall that $4p^2 \leq \log^2 n_h$, and obtain $N_f \leq Ckn_h \log^2 n_h$.

■

Lemma 7.3.3 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the computational complexity of the algorithm "Root to Leaves" has the following upper bound:*

$$N = N_f + N_g \leq Ckn_h \log^2 n_h, \quad C \in \mathbb{R}_+.$$

Proof: See Lemmas 7.3.1, 7.3.2.

Lemma 7.3.4 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the storage of all mappings Φ_ω, $\omega \in T_{\mathcal{T}_h}$, has the following upper bound:*

$$S(\Phi) := S(\Phi^g) + S(\Phi^f) \leq Ckn_h \log^2 n_h.$$

Proof: From $\Phi^g_\omega : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\gamma)}$, $\Phi^f_\omega : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\gamma)}$, $|I(\partial\omega)| \leq |I(\omega)|$ follows that $S(\Phi^g_\omega) \leq S(\Phi^f_\omega)$. Therefore it is enough to estimate only the second term

$$\begin{aligned}
S(\Phi^f) &= \sum_{\omega \in T_{\mathcal{T}_h}} S(\Phi^f_\omega) \\
&= \sum_{i=0}^{2p} 2^i S(\Phi^f_{\omega_i}) \\
&= \sum_{i=0}^{2p} 2^i (C_i k n_h(\omega_i) \log n_h(\omega_i)) \\
&\leq \sum_{i=0}^{2p} 2^i C_i' k \frac{n_h}{2^i} \log \frac{n_h}{2^i} \\
&\overset{C' := \max_i C_i'}{\leq} C' k n_h \sum_{i=0}^{2p} (2p + 1 - i) \\
&\leq C' k n_h p(2p+2) \leq Ckn_h \log^2 n_h, \quad C \in \mathbb{R}_+.
\end{aligned}$$

■

Table 7.1 compares the memory requirement of HDD with the memory requirements of $\mathcal{H}$-Cholesky factorisation and the direct $\mathcal{H}$-matrix inverse. The memory needed for HDD is close to the memory needed for $\mathcal{H}$-Cholesky and much smaller than the memory for the direct $\mathcal{H}$-matrix inverse.

Let us in Lemma 7.3.5 estimate the storage requirements of HDD if only the functionals which compute the mean values in all $\omega \in T_{\mathcal{T}_h}$ are of interest (see Section 4.4.6).

Lemma 7.3.5 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. Let the model domain be as in Remark 7.1.2. HDD requires $S \leq Ckn_h \log n_h$, $C \in \mathbb{R}_+$, units of memory for storing all functionals $\lambda^f_\omega : \mathbb{R}^{I(\omega)} \to \mathbb{R}$ and $\lambda^g_\omega : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}$.*

ε_a	$\mathcal{H}$-Cholesky(MB)	HDD(MB)	$(A^{-1})^{\mathcal{H}}$(MB)
10^{-3}	13.3	19.7	51.0
10^{-4}	14.7	20.1	64.0
10^{-5}	16.0	20.4	75.2
10^{-6}	17.2	20.6	87.4

Table 7.1: Dependence of memory requirements on the adaptive rank arithmetic parameter ε_a, where rank $k = \operatorname{argmin}\{\sigma_k \le \varepsilon_a \sigma_1\}$. The number of degrees of freedom is 129^2.

Proof: Since the functional λ^f_ω requires more resources than λ^g_ω, we, therefore, perform estimates only for λ^f_ω:

$$\begin{aligned} S &= \sum_{\omega \in T_{\mathcal{T}_h}} S(\lambda^f_\omega) \le \sum_{i=0}^{2p} 2^i \cdot S(\lambda^f_{\omega_i}) \\ &= \sum_{i=0}^{2p} 2^i n_h(\omega_i) \le \sum_{i=0}^{2p} 2^i \frac{n_h}{2^i} \\ &= (2p+1) n_h \le C n_h \log n_h, \quad C \in \mathbb{R}_+. \end{aligned}$$

7.4 Modifications of the HDD Method

7.4.1 HDD with Truncation the Small Scales - Case (b)

The algorithm "Leaves to Root" starts with the leaves of $T_{\mathcal{T}_h}$ and goes until the root, while the algorithm "Root to Leaves" starts with the root of $T_{\mathcal{T}_h}$ and stops when $diam(\omega) \le H$, $H \ge h$. The mappings Φ^g_ω and Φ^f_ω are stored only for $\omega \in T^{\ge H}_{\mathcal{T}_h}$ (see Fig. 4.8).

Lemma 7.4.1 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the storage requirements for all mappings Φ^g_ω and Φ^f_ω, $\omega \in T^{\ge H}_{\mathcal{T}_h}$, have the following upper bounds:*

$$S(\Phi^g) \le 42k\sqrt{n_h n_H},$$

$$S(\Phi^f) \le Ck\sqrt{n_h n_H} \log\sqrt{n_h} \log\sqrt{n_H}, \quad C \in \mathbb{R}_+.$$

Proof: The matrix Φ^g_ω, $\omega \in T^{\ge H}_{\mathcal{T}_h}$, is approximated in the low-rank format and belongs to $\mathcal{R}(k, I(\gamma_\omega), I(\partial\omega))$. The storage requirement of Φ^g_ω is less than $k(n_{h,x}(\omega) + 6n_{h,x}(\omega)) = 7k \cdot n_{h,x}(\omega)$. The total storage of all Φ^g_ω, $\omega \in T^{\ge H}_{\mathcal{T}_h}$, can be estimated as

follows

$$\begin{aligned}
S(\Phi^g) &:= \sum_{\omega \in T^{\geq H}_{\mathcal{T}_h}} S(\Phi^g_\omega) \\
&= S(\Phi^g_{\omega_0}) + 2S(\Phi^g_{\omega_1}) + 4S(\Phi^g_{\omega_2}) + 8S(\Phi^g_{\omega_3}) + ... + 2^{2q+1}S(\Phi^g_{\omega_{2q+1}}) \\
&\leq 3S(\Phi^g_{\omega_0}) + 12S(\Phi^g_{\omega_2}) + ... + (2^{2q} + 2^{2q+1})N(\Phi^g_{\omega_{2q}}) \\
&\leq \sum_{i=0}^{q} (2^{2i+1} + 2^{2i})S(\Phi^g_{\omega_{2i}}) \\
&\leq \sum_{i=0}^{q} (2^{2i+1} + 2^{2i})(7k \cdot n_{h,x}(\omega_{2i})) \\
&= 21k \sum_{i=0}^{q} 2^{2i}(\frac{n_{h,x}}{2^i}) \\
&= 21kn_{h,x} \sum_{i=0}^{q} 2^i < 21kn_{h,x}2^{q+1} \\
&\leq 42kn_{h,x}n_{H,x} = 42k\sqrt{n_h n_H}.
\end{aligned}$$

The total storage of all Φ^f_ω, $\omega \in T^{\geq H}_{\mathcal{T}_h}$, can be estimated as follows

$$\begin{aligned}
S(\Phi^f_\omega) &= \sum_{\omega \in T^{\geq H}_{\mathcal{T}_h}} S(\Phi^f_\omega) \\
&= \sum_{i=0}^{2q} 2^i (C_i \cdot 2kn_{h,x}(\omega_i) \cdot n_{H,x}(\omega_i) \log 2n_{h,x}(\omega_i) \cdot n_{H,x}(\omega_i)) \\
&\leq \sum_{i=0}^{2q} 2^i (C'_i 2k \frac{n_{h,x} \cdot n_{H,x}}{2^i} \log(\frac{2n_{h,x} \cdot n_{H,x}}{2^i})) \\
&\overset{C' := \max_i C'_i}{=} 2C'kn_{h,x}n_{H,x} \sum_{i=0}^{2q} (\log(n_{h,x}n_{H,x}) - i + 1) \\
&\leq 2C'kn_{h,x}n_{H,x} \sum_{i=0}^{2q} (p + 1 + q + 1 - i + 1) \\
&\leq 2C'kn_{h,x}n_{H,x}(p+3)(2q+1) \leq Ck\sqrt{n_h n_H} \log \sqrt{n_h} \cdot \log \sqrt{n_H},\ C \in \mathbb{R}_+.
\end{aligned}$$

■

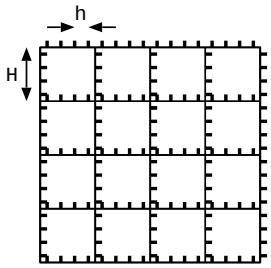


Figure 7.4: An example with two grids. Level $2q$ corresponds to the scale H and level $2p$ to the scale h. The solution in $\omega \in T_{\mathcal{T}_h}$ with $diam(\omega) \geq H$ is of interest.

Lemma 7.4.2 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the domain from Remark 7.1.2 the following two statements hold. a) The computational complexity of all matrix-vector multiplications $\Phi^g_\omega \cdot g_\omega$, where $\Phi^g_\omega : \mathbb{R}^{I(\partial\omega_h)} \to \mathbb{R}^{I(\gamma_h)}$, $\omega \in T^{\geq H}_{\mathcal{T}_h}$, $g_\omega \in \mathbb{R}^{I(\partial\omega_h)}$, has the following upper bound:*

$$N_g \leq 84k\sqrt{n_h n_H}.$$

b) The computational complexity of all matrix-vector multiplications $\Phi^f_\omega \cdot f_\omega$, where $\Phi^f_\omega : \mathbb{R}^{I(\omega_h)} \to \mathbb{R}^{I(\gamma_h)}$, $\omega \in T^{\geq H}_{\mathcal{T}_h}$, $f_\omega \in \mathbb{R}^{I(\omega_h)}$, has the following upper bound:

$$N_f \leq Ck\sqrt{n_h n_H}\log\sqrt{n_h}\log\sqrt{n_H}, \quad C \in \mathbb{R}_+.$$

Proof: Let $R \in \mathcal{R}(k,n,m)$ be a low-rank matrix and $v \in \mathbb{R}^m$ a vector. Recall that the storage $S(R)$ is equal to $k(n+m)$ and the cost N of $R \cdot v$ is less than $2k(n+m)$, i.e., $N(R \cdot v) \leq 2S(R)$. Then Lemma 7.4.1 yields

$$N_g \leq 2 \cdot 42k\sqrt{n_h n_H}.$$

b) Let M be an $\mathcal{H}$-matrix and v a vector. Due to Lemma 5.10.2, $N(M \cdot v) \leq 2S(M)$, where $N(M \cdot v)$ is the cost of the multiplication and $S(M)$ is the storage requirement of M. From Lemma 7.4.1 follows that

$$N_f \leq 2S(\Phi^f) \leq 2C'k\sqrt{n_h n_H}\log\sqrt{n_h}\log\sqrt{n_H}, \quad C' \in \mathbb{R}_+.$$

■

Lemma 7.4.3 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the domain from Remark 7.1.2 the computational complexity of the algorithm "Root to Leaves" on two grids with step sizes h and H has the following upper bound:*

$$N = N_f + N_g \leq Ck\sqrt{n_h n_H}\log\sqrt{n_h}\log\sqrt{n_H}, \quad C \in \mathbb{R}_+.$$

Proof: Use the fact $N_f > N_g$ and Lemma 7.4.2.

7.4.2 HDD on Two Grids - Case (c)

We construct Ψ^g_ω and Φ^g_ω as in Case (a) (Sections 7.2, 7.3). If the right-hand side f is smooth, then it is enough to consider the right-hand side f in a coarse space $V_H \subset V_h$. In this case we construct the mappings $\Psi^f_\omega : \mathbb{R}^{I(\omega_H)} \to \mathbb{R}^{I(\partial\omega_h)}$ and $\Phi^f_\omega : \mathbb{R}^{I(\omega_H)} \to \mathbb{R}^{I(\gamma_h)}$ from $\Psi^f_{\omega_i} : \mathbb{R}^{I(\omega_{i,H})} \to \mathbb{R}^{I(\partial\omega_{i,h})}$, $i = 1, 2$, $\omega, \omega_1, \omega_2 \in T_{\mathcal{T}_h}$.

Remark 7.4.1 *Note that in this modification of HDD the matrices Ψ^f_ω, Φ^f_ω have a smaller number of columns ($|I(\omega_H)| \leq |I(\omega_h)|$) than the respective matrices in Cases (a) and (b). The number of degrees of freedom on the fine grid is $n_h = n_{h,x} n_{h,x}$, the number of degrees of freedom on the interface (see Fig. 7.4) is $n_{h,H} \leq 2 \cdot n_{h,x} n_{H,x}$, where $n_{H,x} \leq n_{h,x}$.*

Lemma 7.4.4 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the computational complexity of all mappings $\Psi^f_\omega : \mathbb{R}^{I(\omega_H)} \to \mathbb{R}^{I(\partial\omega_h)}$, $\omega \in T^{\geq H}_{\mathcal{T}_h}$, has the following upper bound:*

$$N(\Psi^f) \leq Ck^2 \sqrt{n_H n_h} \log^3 \sqrt{n_H n_h}, \quad C \in \mathbb{R}_+.$$

Proof: Due to Lemma 7.2.2, the computational complexity of all mappings Ψ^f_ω, $\omega \in T_{\mathcal{T}_h}$, is estimated by $C' k^2 n_h \log^3 n_h$. On the interface (see Fig. 7.4) the number of nodal points is $2\sqrt{n_h n_H}$. We substitute n_h in $C' k^2 n_h \log^3 n_h$ by $2\sqrt{n_h n_H}$ and obtain

$$N(\Psi^f) \leq C' k^2 \cdot 2\sqrt{n_h n_H} \log^3(2\sqrt{n_h n_H}) \leq Ck^2 \sqrt{n_H n_h} \log^3 \sqrt{n_H n_h}.$$

■

Lemma 7.4.5 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the storage of all mappings $\Phi^f_\omega : \mathbb{R}^{I(\omega_H)} \to \mathbb{R}^{I(\gamma_h)}$, $\omega \in T^{\geq H}_{\mathcal{T}_h}$, has the following upper bound:*

$$S(\Phi^f) \leq Ck \sqrt{n_H n_h} \log^2 \sqrt{n_H n_h}, \quad C \in \mathbb{R}_+.$$

Proof: From Lemma 7.3.4 follows that the storage requirement for all mappings Φ^f_ω, $\omega \in T_{\mathcal{T}_h}$, is estimated by $C' k n_h \log^2 n_h$, $C' \in \mathbb{R}_+$. Now the number of nodal points is not n_h, but $2\sqrt{n_h n_H}$. Therefore

$$S(\Phi^f_\omega) \leq C' k \cdot 2\sqrt{n_h n_H} \log^2(2\sqrt{n_h n_H}) \leq Ck \sqrt{n_h n_H} \log^2 \sqrt{n_h n_H}.$$

■

In Table 7.2 we present the dependences of the storage requirements for the mappings Φ^g, Φ^f on h and H (domain Ω as in Remark 7.1.2). Here

$$\|\mathbf{u} - \tilde{\mathbf{u}}_k\|_2 \,/\, \|\mathbf{u}\|_2 \approx 10^{-5}, \quad \|\mathbf{u} - \tilde{\mathbf{u}}_k\|_\infty \approx 10^{-5},$$

where $\mathbf{u}$ is the exact solution and $\tilde{\mathbf{u}}_k$ the approximated solution. We see that Φ^g has a linear memory requirement and Φ^f an almost linear memory requirement. The memory requirement of HDD on two grids with step sizes $H = 0.5$ and h is in a factor ~ 2 smaller (the third column) than the memory requirements of the standard HDD (the first column). For $H = 0.125$ the factor is ~ 1.4 (the forth column).

h	Φ^g,Φ^f,$H = h$,kB	Φ^g,Φ^f,$H = 0.5$,kB	Φ^g,Φ^f,$H = 0.125$,kB
1/33	$2.45 * 10^2$, $4 * 10^2$	$9.1 * 10$, $1.7 * 10^2$	$2 * 10^2$, $2.8 * 10^2$
1/65	$1.1 * 10^3$, $2.4 * 10^3$	$2.9 * 10^2$, $1.2 * 10^3$	$7.9 * 10^2$, $1.8 * 10^3$
1/129	$5 * 10^3$, $1.4 * 10^4$	$6.8 * 10^2$, $8 * 10^3$	$2.6 * 10^3$, $1.2 * 10^4$
1/256	$2.1 * 10^4$, $7.86 * 10^4$	$1.4 * 10^3$, $4.1 * 10^4$	$7.4 * 10^3$, $6.9 * 10^4$

Table 7.2: The dependence of the memory requirements for Φ^g and Φ^f on the grid step sizes h, H.

7.4.3 HDD on Two Grids and with Truncation of Small Scales - Case (d)

This case combines Cases (b) and (c). Suppose that the right-hand side f is given in the space $V_H \subset V_h$. We prolongate $f_H \in V_H$ onto the fine space V_h and obtain $f_h := P_{h \leftarrow H} f_H$. After that we construct the mappings $\Psi^f_\omega : \mathbb{R}^{I(\omega_H)} \rightarrow \mathbb{R}^{I(\partial\omega_h)}$ and $\Phi^f_\omega : \mathbb{R}^{I(\omega_H)} \rightarrow \mathbb{R}^{I(\gamma_h)}$ for all $\omega \in T^{\geq H}_{\mathcal{T}_h}$.

Lemma 7.4.6 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the computational complexity of the algorithm "Leaves to Root" has the following upper bound:*

$$N(\Psi) = N(\Psi^f_\omega) + N(\Psi^g_\omega) \leq Ck^2 \sqrt{n_H n_h} \log^3 \sqrt{n_h n_H}, \quad C \in \mathbb{R}_+.$$

Proof: Analogously to Case (c). See the proof of Lemma 7.4.4.

■

Lemma 7.4.7 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the computational complexity of the algorithm "Root to Leaves" has the following upper bound:*

$$N = N_g + N_f \leq Ck \sqrt{n_h n_H} \log \sqrt{n_h} \log \sqrt{n_H}, \quad C \in \mathbb{R}_+.$$

Proof: Analogously to Case (b). See the proof of Lemma 7.4.2.

Lemma 7.4.8 *Let k be the rank which is used in the $\mathcal{H}$-matrix arithmetic. For the model domain from Remark 7.1.2 the storage requirements of the mappings Φ^g_ω and Φ^f_ω for all $\omega \in T^{\geq H}_{\mathcal{T}_h}$ have the following upper bounds:*

$$S(\Phi^g_\omega) \leq 42k \sqrt{n_h n_H} \text{ and}$$

$$S(\Phi^f_\omega) \leq Ck \sqrt{n_h n_H} \log \sqrt{n_h} \log \sqrt{n_H}.$$

Proof: These storage requirements are the same as in Case (b). See the proof of Lemma 7.4.1.

8 Parallel Computing

In this chapter we present the parallel HDD algorithm and estimate its complexity.

8.1 Introduction

In this chapter we consider the parallel computers with the shared memory architecture (although the distributed memory architecture is also possible). We will consider the parallel RAM (PRAM) model, which consists of q processors and a global memory which can be accessed simultaneously by all processors. All data transfers between different processors are handled by this memory system. The communication time between processors is negligible small in comparison with the computational time.
To characterize parallel algorithms we introduce the following notation:

Definition 8.1.1 *Let $t(q)$ be the execution time of the parallel algorithm $\mathcal{A}$ on a machine with q processors. Then we denote the parallel speedup by $S(q) := \frac{t(1)}{t(q)}$ and the parallel efficiency of $\mathcal{A}$ by $E(q) := \frac{S(q)}{q} = \frac{t(1)}{q \cdot t(q)}$.*

To parallelize the HDD method one needs to do the following:

1. Perform the triangulation $\mathcal{T}_h$ of the domain Ω and construct the hierarchical decomposition tree $T_{\mathcal{T}_h}$ in parallel.

2. Perform the parallelization of the $\mathcal{H}$-matrix arithmetic (described by R.Kriemann in his dissertation [44]).

3. Perform the parallelization of the algorithm "Leaves to Root".

4. Perform the parallelization of the algorithm "Root to Leaves" (requires the parallel $\mathcal{H}$-matrix-vector multiplication).

The time for item (1) is much smaller than the time needed for items (3) and (4). The parallel $\mathcal{H}$-matrix arithmetic is used in items (3) and (4).

8.2 Parallel Algorithms for $\mathcal{H}$-Matrix Arithmetics

The parallel algorithms for the matrix-vector multiplication, matrix-matrix addition, matrix-matrix multiplication and LU-decomposition for dense matrices can be found in [25]. The parallel versions of the respective algorithms for $\mathcal{H}$-matrices can be found in [44].

Theorem 8.2.1 *Let $T := T_{I\times I}$ be a block cluster trees, $M, M', M'' \in \mathcal{H}(T_{I\times I}, k)$, $n := |I|$, $x \in \mathbb{R}^n$. Suppose $n_{min} > k$. Let $|\mathcal{L}(T)|, |V(T)|$ be the numbers of leaves and of nodes in the block cluster tree $T_{I\times I}$, $q = 2^r$ is the number of parallel processors. Then the computational complexities in Table 8.1 hold.*

Operation	Sequential Complexity	Parallel Complexity
$building(M)$	$N = \mathcal{O}(n \log n)$	$\frac{N}{q} + \mathcal{O}(\|V(T)\backslash\mathcal{L}(T)\|)$
$storage(M)$	$N = \mathcal{O}(kn \log n)$	$\frac{N}{q}$
Mx	$N = \mathcal{O}(kn \log n)$	$\frac{N}{q}$
$M' \oplus M''$	$N = \mathcal{O}(k^2 n \log n)$	$\frac{N}{q}$
$M' \odot M''$	$N = \mathcal{O}(k^2 n \log^2 n)$	$\frac{N}{q} + \mathcal{O}(C_{sp}(T)\|V(T)\|)$
M^{-1}	$N = \mathcal{O}(k^2 n \log^2 n)$	$\frac{N}{q} + \mathcal{O}(nn_{min}^2)$
$\mathcal{H}$-LU	$N = \mathcal{O}(k^2 n \log^2 n)$	$\frac{N}{q} + \mathcal{O}(\frac{k^2 n \log^2 n}{n^{1/d}})$

Table 8.1: Computational complexities for sequential and parallel algorithms.

Proof: For the proof of the estimates from the second column see [27] or [33]. For the proof of the estimates from the third column see [45] or [44].

Definition 8.2.1 *Let v be a vertex of $T_{\mathcal{T}_h}$, then we denote the subtree which has vertex v as a root by $T_{\mathcal{T}_h}(v)$. The set of all nodes at the level l of $T_{\mathcal{T}_h}$ we denote by $T_{\mathcal{T}_h}^{(l)}$.*

Remark 8.2.1 *Since the parallel efficiency for small matrices is much smaller than for large matrices, we divide $T_{\mathcal{T}_h}$ into two parts $T_{\mathcal{T}_h}^{(l<r)}$ and $T_{\mathcal{T}_h}^{(l\geq r)}$ (see Fig. 8.1), where $r := \log q$. $T_{\mathcal{T}_h}^{(l<r)}$ contains domains with a large number of degrees of freedom and $T_{\mathcal{T}_h}^{(l\geq r)}$ with a small number. The parallel arithmetic is applied only for levels $0, .., r-1$. The matrices which appear on the levels $r, ..., 2p$ are small enough and are computed by one processor.*

Assumption 8.2.1 *Let ω_1 be a son of ω. We suppose that $|I(\omega_1)| \approx \frac{1}{2}|I(\omega)|$, $|I(\partial\omega_1)| \leq \frac{3}{4}|I(\partial\omega)|$ and $|I(\partial\omega)| \leq 6|I(\gamma_\omega)|$.*

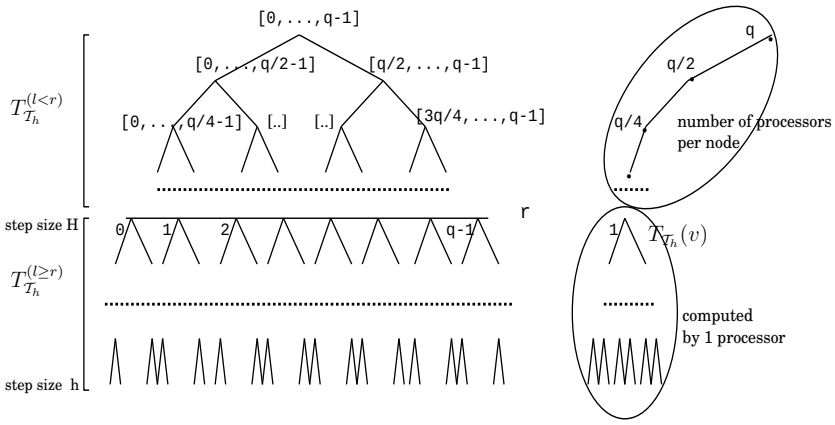


Figure 8.1: **(left)** Decomposition of the $T_{\mathcal{T}_h}$ into $T_{\mathcal{T}_h}^{(l<r)}$ and $T_{\mathcal{T}_h}^{(l\geq r)}$ and distribution of q processors. **(right)** The matrices at the level $l \in [0,..,r-1]$ are computed by $\frac{q}{2^l}$ processors. The subtree $T_{\mathcal{T}_h}(v)$ is computed by one processor.

The parallel version of the algorithm "Leaves to Root"

On each level $l < r$ of $T_{\mathcal{T}_h}$ (the root has level $l = 0$) every node gets 2^{r-l} processors. Every subtree $T_{\mathcal{T}_h}(v)$, $v \in T_{\mathcal{T}_h}^{(r)}$, is computed by one processor (see Fig. 8.1 (right)).

Lemma 8.2.1 *Let $S_\omega := A_{11} - A_{12}A_{22}^{-1}A_{21}$ (see (6.1)), where $A_{11} \in \mathcal{H}(T_{I(\partial\omega)\times I(\partial\omega)}, k)$, $A_{12} \in \mathcal{R}(k, I(\partial\omega), I(\gamma_\omega))$, $A_{21} \in \mathcal{R}(k, I(\gamma_\omega), I(\partial\omega))$, $A_{22} \in \mathcal{H}(T_{I(\gamma_\omega)\times I(\gamma_\omega)}, k)$, $\omega \in T_{\mathcal{T}_h}$, and $n_{h,\gamma} := |I(\gamma_\omega)|$. The parallel complexity of the computation of S_ω has the following upper bound:*

$$N(S_\omega, q) \leq \frac{C_1 k^2 n_{h,\gamma} \log^2 n_{h,\gamma}}{q} + C_2 n_{min}^2 n_{h,\gamma}, \quad C_1, C_2 \in \mathbb{R}_+. \tag{8.1}$$

Proof: Due to Table 5.3 the parallel complexity of the inversion A_{22}^{-1} is

$$\frac{C_1 k^2 n_{h,\gamma} \log^2 n_{h,\gamma}}{q} + C_0 n_{min}^2 n_{h,\gamma}, \quad C_0, C_1 \in \mathbb{R}_+.$$

The parallel complexity of the multiplication $A_{12} \cdot A_{22}^{-1}$ is k-times the MV multiplication, i.e., $\frac{C_2 k^2 n_{h,\gamma} \log n_{h,\gamma}}{q}$, $C_2 \in \mathbb{R}_+$. The parallel complexity of the multiplication $(A_{12}A_{22}^{-1}) \cdot A_{21}$ (product of two low-rank matrices defined in Lemma 5.6.4) is

$$\frac{2k^2}{q}(|I(\partial\omega)| + |I(\gamma_\omega)|) \overset{Assum.\ 8.2.1}{\leq} \frac{14k^2 n_{h,\gamma}}{q}.$$

The parallel complexity of the subtraction $A_{11} - A_{12}A_{22}^{-1} \cdot A_{21}$ is

$$\frac{C_3 k^2 |I(\partial\omega)| \log |I(\partial\omega)|}{q} \overset{Assum.\ 8.2.1}{\leq} \frac{C_4 k^2 n_{h,\gamma} \log n_{h,\gamma}}{q}, \quad C_3, C_4 \in \mathbb{R}_+.$$

Thus,

$$N(S_\omega, q) \leq \frac{1}{q}(C_4 k^2 n_{h,\gamma} \log n_{h,\gamma} + C_1 k^2 n_{h,\gamma} \log^2 n_{h,\gamma}) + C_0 n_{min}^2 n_{h,\gamma}$$

$$\leq \frac{C k^2 n_{h,\gamma} \log^2 n_{h,\gamma}}{q} + C_0 n_{min}^2 n_{h,\gamma}, \quad C_0, C \in \mathbb{R}_+. \tag{8.2}$$

■

Lemma 8.2.2 *Let $Z_\omega := F_1 - A_{12} A_{22}^{-1} F_2$ (see Lemma 6.2.2), where $\omega \in T_{\mathcal{T}_h}$,*

$$A_{12} \in \mathcal{R}(k, I(\partial\omega), I(\gamma_\omega)), \quad A_{22} \in \mathcal{H}(T_{I(\gamma_\omega)\times I(\gamma_\omega)}, k),$$
$$F_1 \in \mathcal{H}(T_{I(\partial\omega)\times I(\omega)}, k), \quad F_2 \in \mathcal{H}(T_{I(\gamma_\omega)\times I(\omega)}, k),$$

$n_h := |I(\omega)|$. Let the model domain be as in Remark 7.1.2. The parallel complexity of the computation of Z_ω on a machine with q processors has the following upper bound:

$$N(Z_\omega, q) \leq \frac{C k^2 n_h \log n_h}{q}, \quad C \in \mathbb{R}_+. \tag{8.3}$$

Proof: $A_{12} \cdot A_{22}^{-1}$ was computed in parallel for Ψ_ω^g (Lemma 8.2.1). The complexity of the multiplication $(A_{12} A_{22}^{-1}) \cdot F_2$ (the product of a low-rank matrix and an $\mathcal{H}$-matrix is k-times the $\mathcal{H}$-matrix-vector multiplication) is estimated by $\frac{C_1 k^2 n_h \log n_h}{q}$, $C_1 \in \mathbb{R}_+$. The complexity of the subtraction $F_1 - A_{12} A_{22}^{-1} F_2$ is $\frac{C_2 k^2 n_h \log n_h}{q}$, $C_2 \in \mathbb{R}_+$. Thus,

$$N(Z_\omega, q) \leq \frac{1}{q}(C_1 k^2 n_h \log n_h + C_2 k^2 n_h \log n_h) \leq \frac{C k^2 n_h \log n_h}{q}, \ C \in \mathbb{R}_+.$$

■

Lemma 8.2.3 *Let $\Psi_\omega^g \in \mathbb{R}^{I(\partial\omega)\times I(\partial\omega)}$, $\Psi_{\omega_i}^f \in \mathbb{R}^{I(\partial\omega_i)\times I(\partial\omega_i)}$, $\omega, \omega_i \in T_{\mathcal{I}_h}$, $i = 1, 2$, $\omega = \omega_1 \cup \omega_2$. For the model domain from Remark 7.1.2 the parallel computational complexity of building Ψ_ω^g from $\Psi_{\omega_1}^g$ and $\Psi_{\omega_2}^g$ has the following upper bound:*

$$N(\Psi^g, q) \leq \frac{C k^2 n_{h,\gamma} \log^2 n_{h,\gamma}}{q} + C_1 n_{h,\gamma} n_{min}^2, \quad C, C_1 \in \mathbb{R}_+, n_{h,\gamma} := |I(\gamma_\omega)|.$$

Proof: Lemma 5.10.8 states that building $\Psi_{\omega_1}^g|^{I(\partial\omega)\times I(\partial\omega)}$, $\Psi_{\omega_2}^g|^{I(\partial\omega)\times I(\partial\omega)}$ and their adding on a machine with one processor costs

$$N(\Psi^g, 1) \leq C_0 k^2 |I(\partial\omega)| \log^2 |I(\partial\omega)| \leq C' k^2 n_{h,\gamma} \log^2 n_{h,\gamma}, \quad C_0, C' \in \mathbb{R}_+.$$

Theorem 8.2.1 states that the parallel complexity of the $\mathcal{H}$-matrix addition is $N(q) = \frac{N(1)}{q}$. From these two facts follows that adding $\Psi_{\omega_1}^g|^{I(\partial\omega)\times I(\partial\omega)} \oplus \Psi_{\omega_2}^g|^{I(\partial\omega)\times I(\partial\omega)}$ costs $\frac{C' k^2 n_{h,\gamma} \log^2 n_{h,\gamma}}{q}$. After the elimination of the unknowns with indices from $I(\gamma_\omega)$, we obtain Ψ_ω^g. By Lemma 8.2.1 this elimination costs

$$\frac{C_2 k^2 n_{h,\gamma} \log^2 n_{h,\gamma}}{q} + C_1 n_{min}^2 n_{h,\gamma}, \quad C_1, C_2 \in \mathbb{R}_+. \tag{8.4}$$

The total complexity is

$$N(\Psi^g, q) \leq \frac{Ck^2 n_{h,\gamma} \log^2 n_{h,\gamma}}{q} + C_1 n_{min}^2 n_{h,\gamma} \text{ with } C = C_2 + C'. \tag{8.5}$$

Lemma 8.2.4 *Let $\Psi^f_\omega \in \mathbb{R}^{I(\partial\omega)\times I(\omega)}$, $\Psi^f_{\omega_i} \in \mathbb{R}^{I(\partial\omega_i)\times I(\omega_i)}$, $\omega, \omega_i \in T_{\mathcal{T}_h}$, $i = 1, 2$, $\omega = \omega_1 \cup \omega_2$ and $n_h := |I(\omega)|$. For the model domain from Remark 7.1.2 the parallel computational complexity of building Ψ^f_ω from $\Psi^f_{\omega_1}$ and $\Psi^f_{\omega_2}$ has the following upper bound:*

$$N(\Psi^f, q) \leq \frac{Ck^2 n_h \log^2 n_h}{q}, \quad C \in \mathbb{R}_+.$$

Proof: In Lemma 5.10.8 we proved that the building of $\Psi^f_{\omega_1}|^{I(\partial\omega)\times I(\omega)}$ and $\Psi^f_{\omega_2}|^{I(\partial\omega)\times I(\omega)}$ and their truncated addition on a machine with one processor cost $N(\Psi^f, 1) \leq C_1 k^2 n_h \log^2 n_h$, $C_1 \in \mathbb{R}_+$. Theorem 8.2.1 states that the parallel complexity of the $\mathcal{H}$-matrix addition is $N(q) = \frac{N(1)}{q}$. From these two facts follows that adding $\Psi^f_{\omega_1}|^{I(\partial\omega)\times I(\omega)} \oplus \Psi^f_{\omega_2}|^{I(\partial\omega)\times I(\omega)}$ costs $\frac{C_1 k^2 n_h \log^2 n_h}{q}$. The computation of Z_ω from Lemma 8.2.2 costs $\frac{C_2 k^2 n_h \log n_h}{q}$, $C_2 \in \mathbb{R}_+$.
Thus, the total complexity is

$$N(\Psi^f, q) \leq \frac{C_1 k^2 n_h \log^2 n_h}{q} + \frac{C_2 k^2 n_h \log n_h}{q} \leq \frac{Ck^2 n_h \log^2 n_h}{q}, \quad C \in \mathbb{R}_+. \tag{8.6}$$

8.3 Parallel Complexity of the HDD Method

8.3.1 Complexity of the Algorithm "Leaves to Root"

Lemma 8.3.1 *Let q be the number of parallel processors. The model domain and the domain decomposition tree $T_{\mathcal{T}_h}$ is the same as in Remark 7.1.2. The parallel computational complexity of all mappings Ψ^g_ω, $\omega \in T_{\mathcal{T}_h}$, has the following upper bound:*

$$N(\Psi^g, q) \leq \frac{3^r}{2^r} \cdot \frac{C' k^2 \sqrt{n_h} \log^2 \sqrt{n_h} + \tilde{C} k^2 n_h}{q} + C''(1 - \frac{3^r}{4^r})\sqrt{n_h} n_{min}^2, \quad C', C'', \tilde{C} \in \mathbb{R}_+.$$

Proof: We decompose $N(\Psi^g, q)$ into two terms (see the reason in Remark 8.2.1)

$$N(\Psi^g, q) = \sum_{l=0}^{r-1} N(\Psi^g_{\omega_l}, \frac{q}{2^l}) + \sum_{\omega \in T_{\mathcal{T}_h}(v)} N(\Psi^g_\omega, 1), \quad v \in T^{(r)}_{\mathcal{T}_h}. \tag{8.7}$$

Note that if $\omega = \omega_1 \cup \omega_2$ then $|I(\partial\omega)| \leq |I(\partial\omega_1)| + |I(\partial\omega_2)|$. The first term can be estimated as follows:

$$\begin{aligned}
\sum_{l=0}^{r-1} N(\Psi^g_{\omega_l}, \frac{q}{2^l}) &\overset{\text{Lem.8.2.3}}{\leq} C_1 \frac{k^2 n_{h,\gamma} \log^2 n_{h,\gamma}}{q} + C_1' n_{h,\gamma} n_{min}^2 \\
&+ C_2 \frac{k^2 \frac{3n_{h,\gamma}}{4} \log^2 \frac{3n_{h,\gamma}}{4}}{\frac{q}{2}} + C_2' \frac{3n_{h,\gamma}}{4} n_{min}^2 + ... \\
&+ C_r \frac{k^2 (\frac{3}{4})^{r-1} n_{h,\gamma} \log^2 (\frac{3}{4})^{r-1} n_{h,\gamma}}{\frac{q}{2^{r-1}}} + C_r' \frac{3^{r-1}}{4^{r-1}} n_{h,\gamma} n_{min}^2 \\
&\leq 2 \cdot (\frac{3^r}{2^r} - 1) \max_{i=1..r} C_i \cdot \frac{k^2 n_{h,\gamma} \log^2 n_{h,\gamma}}{q} + 4 \max_{i=1..r} C_i' \cdot (1 - \frac{3^r}{4^r}) n_{h,\gamma} n_{min}^2 \\
&\leq C' \frac{3^r}{2^r} \cdot \frac{k^2 n_{h,\gamma} \log^2 n_{h,\gamma}}{q} + C''(1 - \frac{3^r}{4^r}) n_{h,\gamma} n_{min}^2,
\end{aligned}$$

where $C' := 2 \max_{i=1..r} C_i$ and $C'' := 4 \max_{i=1..r} C_i'$. Note that for each subtree $T_{\mathcal{T}_h}(v)$, $v \in T^{(r)}_{\mathcal{T}_h}$, we have just one processor and therefore (see Definition 8.2.1, Assumption 8.2.1)

$$\sum_{\omega \in T_{\mathcal{T}_h}(v)} N(\Psi^g_\omega, 1) \overset{Lemma\ 7.2.1}{\leq} \tilde{C} k^2 \frac{3^r}{4^r} n_h,$$

where $\frac{3^r}{4^r} n_h$ is the number of nodal points in the root of $T_{\mathcal{T}_h}(v)$, $\tilde{C} \in \mathbb{R}_+$. Taking into account the facts $q = 2^r$, $n_{h,\gamma} \leq \sqrt{n_h}$, we obtain

$$N(\Psi^g, q) \leq \frac{3^r}{2^r} \cdot \frac{C' k^2 \sqrt{n_h} \log^2 \sqrt{n_h} + \tilde{C} k^2 n_h}{q} + C''(1 - \frac{3^r}{4^r}) \sqrt{n_h} n_{min}^2.$$

■

Lemma 8.3.2 *Let q be the number of parallel processors. The model domain and the domain decomposition tree $T_{\mathcal{T}_h}$ is the same as in Remark 7.1.2. The parallel computational complexity of all mappings Ψ^f_ω, $\omega \in T_{\mathcal{T}_h}$, has the following upper bound:*

$$N(\Psi^f, q) \leq C k^2 \frac{n_h}{q} \log^3 \frac{n_h}{q}, \quad C \in \mathbb{R}_+.$$

Proof: We decompose $N(\Psi^f, q)$ into two terms (see the reason in Remark 8.2.1)

$$N(\Psi^f, q) = \sum_{l=0}^{r-1} N(\Psi^f_{\omega_l}, \frac{q}{2^l}) + \sum_{\omega \in T_{\mathcal{T}_h}(v)} N(\Psi^f_\omega, 1), \quad v \in T^{(r)}_{\mathcal{T}_h}. \tag{8.8}$$

Recall that $|I(\omega_1)| \approx \frac{1}{2}|I(\omega)|$. The first term can be estimated as follows:

$$\sum_{l=0}^{r-1} N(\Psi^f_{\omega_l}, \frac{q}{2^l}) \overset{\text{Lem.8.2.4}}{\leq} C_1 \frac{k^2 n_h \log^2 n_h}{q} + C_2 \frac{k^2 \frac{n_h}{2} \log^2 \frac{n_h}{2}}{\frac{q}{2}} + ... + C_r \frac{k^2 \frac{n_h}{2^{r-1}} \log^2 \frac{n_h}{2^{r-1}}}{\frac{q}{2^{r-1}}}$$

$$\tilde{C} = \max_{i=1..r} C_i \leq \frac{\tilde{C} k^2 n_h}{q} (\log^2 n_h + \log^2 \frac{n_h}{2} + ... + \log^2 \frac{n_h}{2^{r-1}})$$

$$\leq \frac{\tilde{C} k^2 n_h}{q} ((2p+1)^2 + (2p)^2 + ... + (2p-r+2)^2)$$

$$\leq \frac{\tilde{C} k^2 n_h}{q} (\frac{1}{3}((2p+1)^3 - (2p-r+1)^3)$$

$$\leq \frac{\tilde{C} k^2 n_h}{3q} \cdot (4(2p+1)^2 r - 4(2p+1)r^2 + r^3)$$

$$\leq \frac{C' k^2 n_h (\log n_h - \log q) \log n_h \cdot \log q}{q}.$$

Note that for each $\omega \in T_{\mathcal{T}_h}(v)$ we have just one processor and therefore (see Definition 8.2.1 and Assumption 8.2.1)

$$\sum_{\omega \in T_{\mathcal{T}_h}(v)} N(\Psi^f_\omega, 1) \overset{\text{Lem. 7.2.2}}{\leq} C k^2 n_h(v) \log^3 n_h(v)$$

$$\leq C k^2 \frac{n_h}{2^r} \log^3 \frac{n_h}{2^r}, \quad C \in \mathbb{R}_+.$$

We take into account the following inequality

$$\frac{C' k^2 n_h (\log n_h - \log q) \log n_h \cdot \log q}{q} \leq C_2 k^2 \frac{n_h}{q} \log^3 \frac{n_h}{q}$$

and obtain the statement of the lemma.

■

Lemma 8.3.3 *Let q be the number of parallel processors. The model domain and the domain decomposition tree $T_{\mathcal{T}_h}$ is the same as in Remark 7.1.2. The parallel computational complexity of all mappings Ψ_ω, $\omega \in T_{\mathcal{T}_h}$, has the following upper bound:*

$$N(\Psi, q) \leq \frac{C' k^2 \sqrt{n_h} \log^2 \sqrt{n_h} + \tilde{C} k^2 n_h}{q^{0.45}} + C''(1 - \frac{3^r}{4^r}) \sqrt{n_h} n^2_{min} + C k^2 \frac{n_h}{q} \log^3 \frac{n_h}{q},$$

where $\tilde{C}, C', C'', C \in \mathbb{R}_+$.

Proof: Using Lemmas 8.3.1, 8.3.2, compute $N(\Psi, q) = N(\Psi^f, q) + N(\Psi^g, q)$. Note that $\frac{3^r}{4^r} \approx \frac{1}{q^{0.45}}$ and $n_{h,\gamma} \leq \sqrt{n_h}$.

8.3.2 Complexity of Algorithm "Root to Leaves"

The input data for this algorithm is a set of mappings Φ_ω^g and Φ_ω^f for all $\omega \in T_{\mathcal{T}_h}$. The algorithm "Root to Leaves" performs the multiplications $\Phi_\omega^g \cdot g_\omega$ and $\Phi_\omega^f \cdot f_\omega$.

Notation 8.3.1 *The cost of the multiplication $\Phi_\omega^g \cdot g_\omega$ on a computer with q processors we denote by $N_g(\omega, q)$. The cost of all multiplications $\Phi_\omega^g \cdot g_\omega$, $\omega \in T_{\mathcal{T}_h}$, on a computer with q processors we denote by $N_g(q)$. Similarly, we define $N_f(\omega, q)$ and $N_f(q)$.*

Lemma 8.3.4 *Let q be the number of parallel processors. The model domain and the domain decomposition tree $T_{\mathcal{T}_h}$ is the same as in Remark 7.1.2. The parallel computational complexity of all matrix-vector products $\Phi_\omega^g \cdot g_\omega$, where $\Phi_\omega^g : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\gamma_\omega)}$, $g_\omega \in \mathbb{R}^{I(\partial\omega)}$, $\omega \in T_{\mathcal{T}_h}$, has the following upper bound:*

$$N_g(q) \leq \frac{28k\sqrt{n_h}}{q}\frac{3^r}{2^r} + \frac{21kn_h}{q}.$$

Proof: We divide $N_g(q)$ into two terms (see the reason in Remark 8.2.1)

$$N_g(q) = \sum_{l=0}^{r-1} N_g(\omega_l, \frac{q}{2^l}) + \sum_{\omega \in T_{\mathcal{T}_h}(v)} N_g(\omega, 1), \text{ where } v \in T_{\mathcal{T}_h}^{(r)}. \tag{8.9}$$

Φ_ω^g is approximated in the low-rank format and belongs to $\mathcal{R}(k, I(\gamma_\omega), I(\partial\omega))$. Assumption 8.2.1 gives $|I(\partial\omega)| \leq 6n_{h,x}$, where $|I(\gamma_\omega)| = n_{h,\gamma} \leq n_{h,x} \simeq \sqrt{n}$. The parallel complexity of the multiplication $\Phi_\omega^g \cdot g_\omega$ can be estimated as follows:

$$\frac{2k \cdot (|I(\partial\omega)| + |I(\gamma_\omega)|)}{q} \leq \frac{2k \cdot (6n_{h,x} + n_{h,x})}{q} = \frac{14kn_{h,x}(\omega)}{q}. \tag{8.10}$$

Suppose that the number of external boundary points decreases by a factor $\frac{3}{4}$ after each division, i.e., $|I(\partial\omega_1)| \leq \frac{3}{4}|I(\partial\omega)|$. Taken into account (8.10), the first term in (8.9) can be estimated as follows:

$$\begin{aligned}\sum_{l=0}^{r-1} N_g(\omega_l, \frac{q}{2^l}) &\leq \frac{14kn_{h,x}}{q} + \frac{14k\frac{3n_{h,x}}{4}}{\frac{q}{2}} + ... + \frac{14k\frac{3^{r-1}n_{h,x}}{4^{r-1}}}{\frac{q}{2^{r-1}}} \\ &= 14k\frac{n_{h,x}}{q}(1 + \frac{3}{2} + ... + \frac{3^{r-1}}{2^{r-1}}) \\ &\overset{q=2^r}{\leq} \frac{28k\sqrt{n_h}}{q}(\frac{3^r}{2^r} - 1) \leq \frac{28k\sqrt{n_h}}{q}\frac{3^r}{2^r}.\end{aligned}$$

The second term in (8.9) can be estimated by Lemma 7.3.1 as follows

$$\sum_{\omega \in T_{\mathcal{T}_h}(v)} N_g(\omega, 1) \leq \frac{21kn_h}{q},$$

where $\frac{n_h}{q}$ is the number of degrees of freedom in the root of $T_{\mathcal{T}_h}(v)$.

Lemma 8.3.5 *Let q be the number of parallel processors. The model domain and the domain decomposition tree $T_{\mathcal{T}_h}$ is the same as in Remark 7.1.2. The parallel computational complexity of the matrix-vector products $\Phi^f_\omega \cdot f_\omega$, where $\Phi^f_\omega : \mathbb{R}^{I(\omega)} \to \mathbb{R}^{I(\gamma_\omega)}$, $f_\omega \in \mathbb{R}^{I(\omega)}$, $\omega \in T_{\mathcal{T}_h}$, has the following upper bound:*

$$N_f(q) \le Ck^2 \frac{n_h}{q} \log^2 \frac{n_h}{q}, \quad C \in \mathbb{R}_+.$$

Proof: Each level $i \in [0, 2p-3]$ of the tree $T_{\mathcal{T}_h}$ contains 2^i matrices and every matrix is to be multiplied by vector f. The parallel arithmetic is applied only for levels $0, .., r-1$ (see Figure 8.1). The matrices which appear on the levels $l \ge r$ are small enough and are computed by one processor. Suppose that $|I(\omega_1)| \le \frac{1}{2}|I(\omega)|$ (see Assumption 8.2.1). We divide $N_f(q)$ into two parts

$$N_f(q) = \sum_{l=0}^{r-1} N_f(\omega_l, \frac{q}{2^l}) + \sum_{\omega \in T_{\mathcal{T}_h}(v)} N_f(\omega, 1), \text{ where } v \in T^{(r)}_{\mathcal{T}_h}. \tag{8.11}$$

The first term in (8.11) can be estimated as follows

$$\begin{aligned}
\sum_{l=0}^{r-1} N_f(\omega_l, \frac{q}{2^l}) &\overset{Lem.7.3.2}{\le} C_1 \frac{k^2 n_h \log n_h}{q} + C_2 \frac{k^2 \frac{n_h}{2} \log \frac{n_h}{2}}{\frac{q}{2}} + ... + C_r \frac{k^2 \frac{n_h}{2^{r-1}} \log \frac{n_h}{2^{r-1}}}{\frac{q}{2^{r-1}}} \\
&\overset{C' = \max\limits_{i=1..r} C_i}{\le} C' \cdot \frac{k^2 n_h}{q} (2p + 1 + 2p + ... + 2p - r + 2) \\
&\overset{r=\log q}{\le} C' \frac{k^2 n_h ((2p+1)(2p+2) - (2p-r+1)(2p-r+2))}{2q} \\
&\le \tilde{C} \frac{k^2 n_h (4pr - r^2)}{2q} \\
&\overset{r=\log q}{\le} \tilde{C} \frac{k^2 n_h (2 \log n_h - \log q) \log q}{q}, \quad \tilde{C} \in \mathbb{R}_+.
\end{aligned}$$

By Lemma 7.3.2 the second term in (8.11) can be estimated as follows

$$\sum_{\omega \in T_{\mathcal{T}_h}(v)} N_f(\omega, 1) \le Ck \frac{n_h}{q} \log^2 \frac{n_h}{q}.$$

We take into account the inequality

$$\tilde{C} \frac{k^2 n_h (2 \log n_h - \log q) \log q}{q} \le C'' k^2 \frac{n_h}{q} \log^2 \frac{n_h}{q}$$

and obtain the statement of the lemma.

Lemma 8.3.6 *Let q be the number of parallel processors. The model domain and the domain decomposition tree $T_{\mathcal{T}_h}$ is the same as in Remark 7.1.2. The parallel computational complexity of the algorithm "Root to Leaves" has the following upper bound:*

$$N(q) = N_f(q) + N_g(q)$$
$$\leq Ck^2 \frac{n_h}{q} \log^2 \frac{n_h}{q} + \frac{28k\sqrt{n_h}}{q} \frac{3^r}{2^r}, \quad C \in \mathbb{R}_+.$$

Proof: See Lemmas 8.3.4, 8.3.5 and use the inequality

$$\frac{21kn_h}{q} \leq Ck^2 \frac{n_h}{q} \log^2 \frac{n_h}{q}.$$

9 Implementation of the HDD package

The result of the implementation is a package of programs which uses the following libraries: HLIB, LAPACK, BLAS, external triangulators (for complex geometry). This implementation is done in C language (ANSI/ISO standard). The hierarchical matrix library HLIB (see [28]) is used for the $\mathcal{H}$-matrix arithmetic. HLIB uses the linear algebra packages LAPACK (see [2]) and BLAS (see [1]) for the standard matrix-vector arithmetic. The scheme of the implementation is shown in Fig. 9.1. See more about the HDD package in [49].

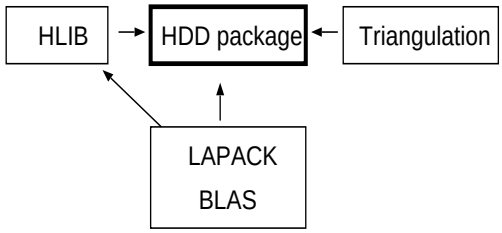


Figure 9.1: The libraries needed for the HDD package.

9.1 Data Structures

Before the application of the hierarchical domain decomposition, a triangulation of the domain Ω has to be made. There are many algorithms which can do this. The triangulation has to satisfy the Definition 3.5.1. The set of vertices with their coordinates is the input data for building a triangulation. The cost of the triangulation algorithm is $\mathcal{O}(n \log n)$ (see [56]). The triangulation includes the following information:

1. List of internal and boundary vertices.
2. List of triangles.
3. For each vertex the list of the adjoint triangles.
4. List of edges.

The HDD method needs a hierarchy of grids. To build this hierarchy we divide each triangle of the coarse grid recursively into four triangles (see Fig. 9.2). We stop the division when all triangles are small enough (see Fig. 9.5). This process provides the hierarchy of grids.

Vertex Structure
index
list of adjoining triangles
property (internal, external)
coordinates[2]

Triangle Structure
index
vertices[3]
edges[3]
property
stiffness matrix[3][3]
father*
sons*

Edge Structure
index
sons*
vertices[2]
property
father*

Table 9.1: Fields of the basic structures (see Remark 9.1.1).

Remark 9.1.1 *The notation* `x`$[N]$ *means that* `x` *is an array, which contains* N *elements. The notation* `y`$*$ *means that* `y` *is either a pointer or an array. The notation* `z`$[N][M]$ *means that matrix* `z` $\in \mathbb{R}^{N\times M}$.

Remark 9.1.2 *Note that the structure* `grid` *contains only pointers to the real vertices, triangles and edges.*

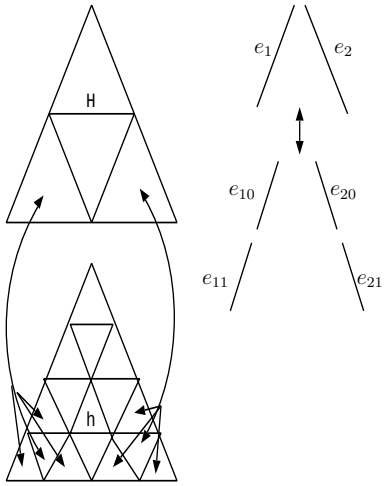


Figure 9.2: Coarse and fine triangulations. Each triangle and each edge of the fine grid contains data about their father.

The structures `vertex`, `triangle` and `edge` are present in Table 9.1. The Diagram in Fig. 9.3 shows the connection between these structures.

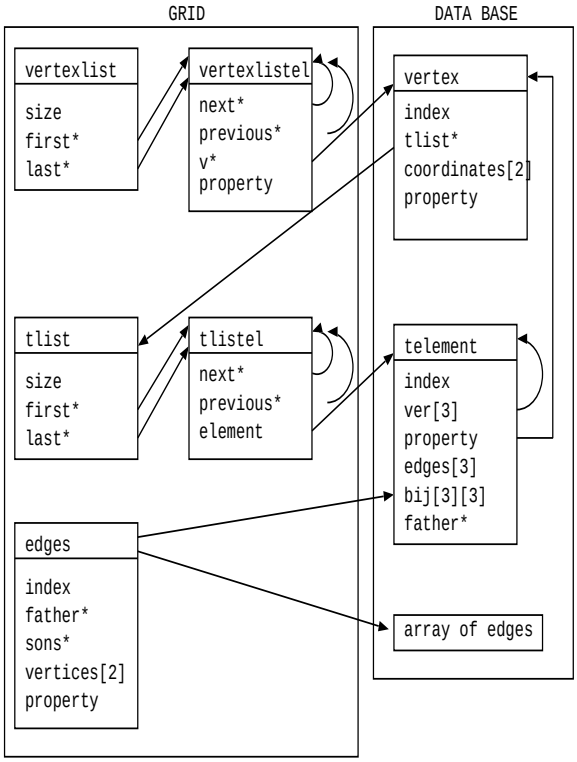


Figure 9.3: Implementation of the structures for storing vertices, triangles and edges.

9.2 Implementation of the Hierarchy of Grids

Until now we had one or two grids. Below we discuss how to implement the hierarchy of grids $\mathcal{T}_h \subset \mathcal{T}_{h/2} \subset ... \subset \mathcal{T}_{h/2^q}$. All grids must be connected with each other. It means that each finite element has to know his father and vice versa. We build a grid $\mathcal{T}_h$ with step size h, refine it, obtain a grid $\mathcal{T}_{h/2}$, refine it again and so on.
In this work we use two grids $\mathcal{T}_{h/2^i}$ and $\mathcal{T}_{h/2^j}$, $0 \leq i, j \leq q$ but for more difficult problems more scales should be used. If we are only interested in two scales with $H/h > 2$, we refine the given scale recursively and do not store intermediate grids. After each recursive step, we reorganize the connections sons↔ father. The scheme of this process is shown in Figures 9.4 and 9.5.

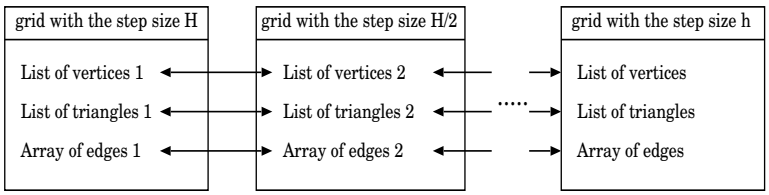


Figure 9.4: Connection of the grids $\mathcal{T}_H$, $\mathcal{T}_{H/2}$,..., $\mathcal{T}_h$.

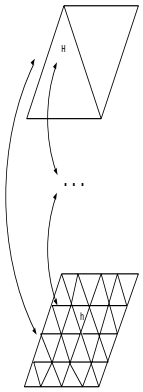


Figure 9.5: All elements of the finest grid have links to the elements of the coarsest grid and vice versa.

Let $\mathcal{T}_{h_0}$ be the initial grid, $\mathcal{T}_H$ the current grid which has to be refined, $\mathcal{T}_h$ the fine grid which we obtain after refinement of $\mathcal{T}_H$. Let i be the index of the current refinement and i_{max} the maximal number of refinements (i.e., $0 \leq i < i_{max}$). Then the algorithm of the recursive mesh refinement will be as follows:

Algorithm 9.2.1 *(Refinement of the grid $\mathcal{T}_H$ i_{max} times)*

```
/* T_H is given, T_{h_0} := T_H, i = 0, i_max ≥ i. */
build_fine_grid(T_{h_0}, T_H, i, i_max)
begin
   T_h:=new_grid();
   if (i = 0) and (i_max = 1) then
      T_h:=refine_grid(T_H);
   else
      if (i < i_max − 1) then
         T'_h:=new_grid();
         T'_h:=refine_grid(T_H);
         copy_links(T_{h_0}, T_H, T'_h);
         delete_grid(T_H);
         build_fine_grid(T_{h_0}, T'_h, T_h, i + 1, i_max);
      else
      if (i = i_max − 1)
         refine_grid(T_H, T_h);
         copy_links(T_{h_0}, T_H, T_h);
         delete_grid(T_H);
      end if;
   end if;
   return T_h;
end
```

Here

- *new_grid()* allocates memory for a new grid.

- *refine_grid*($\mathcal{T}_H$) performs one refinement of $\mathcal{T}_H$ and writes the resulting grid to $\mathcal{T}_h$ (see Algorithm below).
- *copy_links*($\mathcal{T}_{h_0}, \mathcal{T}_H, \mathcal{T}_h$) copies all links "father-to-son" and "son-to-father" from $\mathcal{T}_{h_0}$ to $\mathcal{T}_h$.

Algorithm 9.2.2 *(One step of the mesh refinement procedure)*

refine_grid*(coarse grid $\mathcal{T}_H$)*
begin
  $\mathcal{T}_h$:=*new_grid();*
  for all *vertices of* $\mathcal{T}_H$ ***do***
    add vertex to $\mathcal{T}_h$*;*
  for all *edges of* $\mathcal{T}_H$ ***do***
    add the midpoint of this edge to $\mathcal{T}_h$*;*
    divide the edge into 2 edges and add them to $\mathcal{T}_h$*;*
  end for*;*
  for all *triangles of* $\mathcal{T}_H$ ***do***
    divide triangle in 4 triangles;
    add these 4 triangles to $\mathcal{T}_h$*;*
    add new edges to $\mathcal{T}_h$*;*
  end for*;*
  return $\mathcal{T}_h$*;*
end*;*

9.3 Implementation of the HDD Method

Suppose that the triangulation $\mathcal{T}_h$ of the domain Ω is constructed. In this section we describe how to perform the hierarchical decomposition of Ω (see Fig. 4.1) and implement the HDD method. The scheme of the structures, which are involved in the implementation of the HDD method is shown in Fig. 9.6.
The recursive procedure which performs the hierarchical decomposition is named *divide*(...) (see Algorithm 9.3.1). The input data for the procedure *divide*(...) are the root of $T_{\mathcal{T}_h}$ and the domain Ω. This procedure divides the set of all triangles, the set of all nodes, the external and internal boundary nodes into two parts (left son and right son).
As it was mentioned in Chapter 5, for building an $\mathcal{H}$-matrix we need a cluster tree, but to build this cluster tree we need to know the coordinates of the nodal points $x_i \in \Omega$. At the same time the admissibility condition requires knowledge about the distance between two clusters and diameters of the clusters (see Section 5.4).

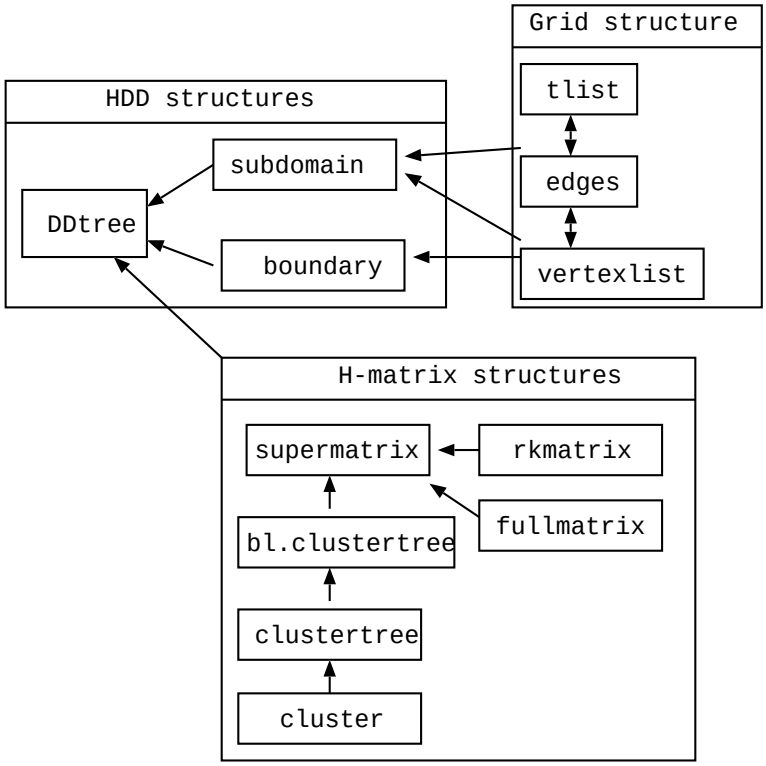


Figure 9.6: The scheme of the structures, which are used for the implementation of the HDD method.

Algorithm 9.3.1 *(Hierarchical decomposition)*

divide*(DDtree* T_ω, subdomain* ω);*
begin
  ω_1 :=*new_domain();*
  ω_2 :=*new_domain();*
  if *(ω contains more than 3 vertices)* ***then***
    divide_vertices(ω, ω_1, ω_2)*;*
    divide_elements(ω, ω_1, ω_2)*;*
    $T_\omega \rightarrow$*leftson := new_DDtree*(ω_1)*;*
    $T_\omega \rightarrow$*rightson := new_DDtree*(ω_2)*;*
    divide($T_\omega \rightarrow$*leftson*, ω_1)*;*
    divide($T_\omega \rightarrow$*rightson*, ω_2)*;*
  else
    $T_\omega \rightarrow$*leftson*=$\varnothing$*;*
    $T_\omega \rightarrow$*rightson*=$\varnothing$*;*
  end if*;*
end*;*

Here

- *divide_vertices*(ω, ω_1, ω_2) divides the set of all nodal points of ω into two subsets $V_1 := \{v|v \in \omega_1\}$ and $V_2 := \{v|v \in \omega_2\}$. Note that $V_1 \cap V_2 = \{v|v \in \gamma_\omega\}$.

- *divide_elements*$(\omega, \omega_1, \omega_2)$ divides the set of all triangles in ω into two subsets $T_1 := \{t | t \in \tau(\omega_1)\}$ and $T_2 := \{t | t \in \tau(\omega_2)\}$. Note that $T_1 \cap T_2 = \varnothing$.

We use the structure `subdomain` to store the data of $\omega \in T_{\mathcal{T}_h}$ (see below). This structure contains the list of all vertices, list of all triangles (on both scales), its boundary box and lists of internal and external boundary vertices (on both scales). The boundary boxes are used for an easier division of a domain into two parts.

To store the data about the interface and the external boundary we use the structure `boundary` (see below). This structure contains the following fields: lists of vertices (`vl`, `cvl`) on both scales and a pointer to the right-hand side matrix (`frhs`). Since each vertex contains the list of joint triangles we can compute its support ((`tl`)).

The main structure for the hierarchical domain decomposition is `DDtree`. This structure combines the information about the current level, the "father" level and its children. `DDtree` is used in both algorithms "Leaves to Root" and "Root to Leaves".

The fields of the structure `DDtree` are shown below. For describing a subdomain $\omega \in T_{\mathcal{T}_h}$ the structure `domain` is used.

Program 9.3.1

```
typedef struct _domain domain;
typedef _domain* pdomain;

struct _domain{
  long        index;  /* Index of the domain */
  tlist*      tl;     /* List of triangles at the fine scale */
  tlist*      ctl;    /* List of triangles at the coarse scale */
  vertexlist* vl;     /* List of vertices at the fine grid */
  vertexlist* cvl;    /* List of vertices at the coarse grid */
  double      area;   /* Area of the domain */
  double      minx,maxx,miny,maxy; /* Describe the boundary box */
}
```

For describing the external $\partial\omega$ and internal γ_ω boundaries the structure `boundary` is used.

Program 9.3.2

```
typedef struct _boundary boundary;
typedef _boundary* pboundary;

struct _boundary{
  vertexlist*  vl;   /* List of vertices at the fine grid   */
  vertexlist*  cvl;  /* List of vertices at the coarse grid */
  tlist*       tl;   /* List of triangles at the fine scale */
  psupermatrix frhs; /* To store the corresponding hierarchical matrix  */
}
```

For describing the HDD tree $T_{\mathcal{T}_h}$ the structure `DDTree` is used.

Program 9.3.3

```
typedef struct _DDTree DDTree;
typedef _DDTree* pDDTree;

struct _DDTree{
  long index;        /* Index of the subdomain *\
  pDomain clus;      /* Pointer to the corresponding domain *\
  pDDTree leftTree;  /* Pointer to the left son *\
  pDDTree rightTree; /* Pointer to the right son  *\
  pDDTree father;    /* Pointer to father *\
  pDDTree brother;   /* Pointer to brother *\

  psupermatrix invA22;
  prkmatrix    phi_g;
  psupermatrix psi;

  double *functional_g, *functional_f;
  int *father2sonL, *father2sonR;
  int  ind_removerow[2], ind_insertrow[2];
  int *dof2idx;
  int  compute;      /* =1 if for this domain matrices are computed, =0 else *\
  int  simple;       /* strategy of building H-matrix (=1 or =2) *\

  pclustertree interct;  /* Cluster tree for the internal boundary (fine grid) */
  pclustertree cinterct; /* Cluster tree for the domain (coarse grid)*/
  pclustertree ect;      /* Cluster tree for the external boundary (fine grid) */
  pclustertree cect;     /* Cluster tree for the external boundary (coarse grid) */
  pclustertree ct;       /* Cluster tree for the domain (fine grid) */
  pclustertree cct;      /* Cluster tree for the domain (coarse grid) */
  pclustertree cl_Gamma; /* Auxiliary cluster tree */
  pclustertree cl_gamma; /* Auxiliary cluster tree */

  pdomain   clus;        /* Pointer to the domain*/
  pboundary eclus;       /* Pointer to the external boundary */
  pboundary interclus;   /* Pointer to the internal boundary */
  int       *cf_index;   /* Auxiliary array. Used for the mesh refinement*/

}
```

To store the inverse of the mapping $\Psi_\omega^g|_{I(\gamma)} : \mathbb{R}^{I(\gamma)} \to \mathbb{R}^{I(\gamma)}$ the field `invA22` is used, to store the mapping $\Phi_\omega^g : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\gamma)}$ the field `phi_g` is used. To store the mapping $\Psi_\omega^g : \mathbb{R}^{I(\partial\omega)} \to \mathbb{R}^{I(\partial\omega)}$ the field `psi` is used. The fields `functional_g`, `functional_f` are needed to store the functionals λ_ω^g and λ_ω^f. The fields `father2sonL`, `father2sonR` are used for storing the mappings $I(\omega) \to I(\omega_1)$ and $I(\omega) \to I(\omega_2)$. The fields `ind_removerow[2]`, `ind_insertrow[2]` store indices from $I(\partial\omega)$ and define which rows should be removed from an $\mathcal{H}$-matrix. The field `dof2idx` maps the set of degrees of freedom on $\partial\omega \cup \gamma$ to the set of indices $I(\partial\omega \cup \gamma)$.

9.4 Conclusion

The main steps of the implementation of HDD are:

1. Read the coarse grid $\mathcal{T}_H$.
2. Refine $\mathcal{T}_H$ by Algorithm 9.2.1 i_{max} times.
3. Build the HDD tree $T_{\mathcal{T}_h}$ by Algorithm 9.3.1.
4. Execute "Leaves to Root" by Algorithm 7.2.1. For each $\omega \in T_{\mathcal{T}_h}$
   a) build Ψ^g_ω and Ψ^f_ω for leaves of $T_{\mathcal{T}_h}$,
   b) build Ψ^g_ω from $\Psi^g_{\omega_1}$ and $\Psi^g_{\omega_2}$ by Algorithms 6.3.1 and 6.3.2,
   c) build Ψ^f_ω from $\Psi^f_{\omega_1}$ and $\Psi^f_{\omega_2}$ by Algorithms 6.3.3 and 6.3.4,
   d) build Φ^g_ω and Φ^f_ω by Algorithm 7.3.1,
   e) compute the functionals λ^g_ω, λ^f_ω by Algorithms 4.4.1 and 4.4.2.
5. Execute "Root to Leaves" by Algorithm 7.3.1:
   a) compute $u_{\gamma_\omega} := \Phi^g_\omega \cdot g_\omega + \Phi^f_\omega \cdot f_\omega$, $\omega \in T_{\mathcal{T}_h}$,
   b) compute (e.g., the mean value) $\lambda_\omega(d_\omega) = (\lambda^f_\omega, f_\omega) + (\lambda^g_\omega, g_\omega)$, $\omega \in T_{\mathcal{T}_h}$.
6. Compute solution by the PCG method and compare it with the solution computed by HDD.

The following modifications of the HDD method were implemented:

1. HDD_1 works with the right-hand side from $V_H \subset V_h$. For this modification the prolongation matrix $P_{h \leftarrow H}$ was applied.
2. HDD_2 computes the solution on all internal boundaries γ_ω, $diam(\omega) \geq H$, and the mean value of the solution inside all domains ω with $diam(\omega) < H$. In this modification the algorithm "Root to Leaves" works only for domains with $diam(\omega) \geq H$.
3. HDD_3 is a combination of HDD_1 and HDD_2.
4. HDD_4 for problems with the homogeneous right-hand side (the mappings Ψ^f_ω and Φ^f_ω are not computed at all).
5. HDD_5 for problems with periodic coefficients (see Section 4.4.4).

10 Numerical Results

Numerical experiments are used for the confirmation of the theoretical results and for the discovering of the invisible patterns of relationships.

In this Chapter we

- compare the solution, obtained by the HDD method, with the solutions obtained by the PCG-method and by the $\mathcal{H}$-Cholesky factorization (see Tables 10.12, 10.13, 10.15, 10.16);
- demonstrate the dependence of the solution u_h on the maximal $\mathcal{H}$-matrix rank k (see Tables 10.5, 10.6, 10.8).
- research the needed computational time and the storage requirement (see Tables 10.14, 10.19, 10.20);
- show the accuracy of the solution for different oscillatory coefficients (see Tables 10.9, 10.10, 10.11, 10.17, 10.18);
- research the accuracy of the $\mathcal{H}$-matrix approximation of Φ^g_ω and Φ^f_ω on different levels of the tree $T_{\mathcal{T}_h}$ (Tables 10.21, 10.22);
- show the absolute and relative errors for the solution of the so-called skin problem with discontinuous coefficients (see Tables 10.25, 10.26, 10.27);
- solve a problem with many right-hand sides by the HDD method and compare the computational time with the time required by PCG (see Table 10.28).

10.1 Notation

Let us introduce the following notation:

- u is an analytic solution;
- $\mathbf{u}$ is the FE exact solution, $\mathbf{u}_i = u(\mathbf{x}_i)$;
- Ω is the model domain and the model grid is shown in Fig. 10.1;
- A is the global stiffness matrix computed for Ω;
- $\mathbf{c}$ is the discrete right-hand side, $A\mathbf{u} = \mathbf{c}$;
- k is the maximal rank used in Definition 5.7.2 of $\mathcal{H}$-matrices. If the rank k in $\mathcal{H}(T_{I\times J}, k)$ is fixed then we call this rank **the $\mathcal{H}$-matrix rank**;

- $\tilde{\mathbf{u}}_k$ is the solution obtained by the HDD method. The subindex k indicates that the fixed rank arithmetic is used (see Def. 5.9.3);
- $\tilde{\mathbf{u}}_\varepsilon$ is the solution obtained by the HDD method. The subindex ε indicates that the adaptive rank arithmetic is used (see Def. 5.9.3);
- $\tilde{\mathbf{u}}_L$ is the solution obtained by the $\mathcal{H}$-Cholesky decomposition ($A\mathbf{u} = \mathbf{c} \Rightarrow \tilde{\mathbf{u}}_L = (LL^T)^{-1}\mathbf{c}$);
- ε_{cg} is the value which is used for the stopping criterium for PCG, i.e., PCG stops as soon as $\|A\tilde{\mathbf{u}}^{(m)} - \mathbf{c}\|_2 \leq \varepsilon_{cg}$;
- $\tilde{\mathbf{u}}_{cg}$ is the solution obtained by the PCG method (with the $\mathcal{H}$-Cholesky preconditioner). As stopping criterium we use the residual $\|A\tilde{\mathbf{u}}_{cg} - \mathbf{c}\|_2 \leq \varepsilon_{cg}$;
- ε_a is the parameter for the adaptive rank arithmetic with the property $k = \min\{i : \sigma_i \leq \varepsilon_a \sigma_1\}$, where σ_i is the i-th singular value;
- ε_h is the discretisation error;
- $\varepsilon_{\mathcal{H}}$ is the $\mathcal{H}$-matrix approximation error;
- n_{min} is the minimal size of an inadmissible block. n_{min} tells us when we should stop divide subblocks further (see Section 5.5.2). By default, the minimal size $n_{min} = 32$ is used for inadmissible blocks;
- $A^{-\mathcal{H}}$ is the $\mathcal{H}$-matrix approximant to the inverse of A.

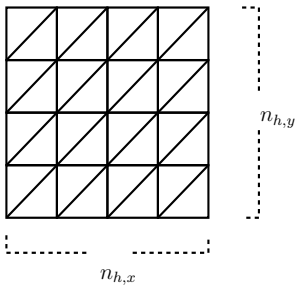


Figure 10.1: The model grid on $\Omega = (0,1)^2$. The number of degrees of freedom is $n_{h,x} \cdot n_{h,y}$, $n_{h,x} = n_{h,y}$.

The model elliptic boundary value problem to be solved is

$$\begin{array}{ll} -\operatorname{div}(\alpha(\mathbf{x})\nabla u) = f & \text{in } \Omega = (0,1)^2, \\ u = g & \text{on } \partial\Omega. \end{array} \tag{10.1}$$

Further, in all experiments, we will use the grid shown in Fig. 10.1.

All numerical experiments were performed on the computers from Table 10.1. The notebook was used for small applications and Kepler for large ones. In order to compute a double integral in a triangle we apply the 12 points quadrature rule (see Table (3.2)).

Name	Model	OS	Speed, Ghz	Memory, Gb
Notebook	Celeron M	Fedora	1.3-1.9	0.5
Kepler	Intel PC	Linux 2.4	3.0	2.0

Table 10.1: Used computers.

We use the domain decomposition as in Proposition 7.1.2. The error $\varepsilon = \|\mathbf{u} - \tilde{\mathbf{u}}_k\|_2$ (or $\varepsilon = \|\mathbf{u} - \tilde{\mathbf{u}}_k\|_\infty$) which we obtain after applying the HDD method is contributed by two errors: the discretization error ε_h and the $\mathcal{H}$-matrix approximation error $\varepsilon_{\mathcal{H}}$. From the definition of $\mathcal{H}$-matrices it is clear that $\varepsilon_{\mathcal{H}} \longrightarrow 0$, when the rank k is increasing. In the standard case the discretisation error is $\varepsilon_h = \mathcal{O}(h)$ (see Section 3.5.2). It is sensible to take the maximal rank k for $\mathcal{H}$-matrices in a way to have the same order of errors ε_h and $\varepsilon_{\mathcal{H}}$. There is no sense to take larger rank k (or smaller ε_a for the adaptive rank arithmetic) to decrease $\varepsilon_{\mathcal{H}}$, because ε_h stays the same and will dominate.

Remark 10.1.1 *Note that to approximate a weak admissible block (see Section 5.5.2) we use the rank* $3 \cdot k$.

Remark 10.1.2 *Contrary to the fixed rank arithmetic, in the adaptive rank arithmetic the needed rank* k *is chosen as follows* $k = \min\{i : \sigma_i \leq \varepsilon_a \sigma_1\}$.

10.2 Preconditioned Conjugate Gradient Method

Usually, the exact solution of the problem (10.1) is unknown. We use the PCG method to estimate the accuracy of the HDD method.

Lemma 10.2.1 *The minimal and maximal singular values of the stiffness matrix of the Poisson-model problem on the rectangular quasi-uniform grid (Fig. 10.1) are:*

$$\lambda_{min} = 8h^{-2} sin^2(\pi h/2),$$
$$\lambda_{max} = 8h^{-2} cos^2(\pi h/2).$$

The matrix A is positive definite and the condition number is

$$cond(A) = \|A\|_2 \|A^{-1}\|_2 = \frac{\lambda_{max}}{\lambda_{min}} \quad or$$

$$cond(A) = \frac{cos^2(\pi h/2)}{sin^2(\pi h/2)} = O\left(\frac{1}{h^2}\right).$$

The discontinuous coefficients $\alpha(\mathbf{x})$ of the operator $\mathrm{div}(\alpha(\mathbf{x})\nabla)$ for the domain as in Fig. 10.8 can increase the condition number of the stiffness matrix dramatically and as a consequence increase the number of iterations. To estimate the upper bound of the condition number one can use the following inequality ([26]):

$$cond(A) \leq \max_{i,j} \left(\frac{\alpha(\triangle_i)}{\alpha(\triangle_j)}\right) h^{-2},$$

where $\triangle_i$, $\triangle_j \in \mathcal{T}_h$.

Table 10.2 shows the dependence of the condition number of the operator $\mathrm{div}(\alpha(\mathbf{x})\nabla)$ (for domain as in Fig. 10.8) on the jumping coefficient α. One can see an exponential increasing of $cond(A)$ with decreasing of α. Table 10.3 shows the dependence of the condition number of the operator $\mathrm{div}(\alpha(\mathbf{x})\nabla)$ on the number of degrees of freedom.

Remark 10.2.1 *To compute the condition number of A we need the minimal and the maximal eigenvalues. It is easy to estimate the maximal eigenvalue and difficult the minimal eigenvalue. For small matrices (Tables 10.2, 10.3) we compute the eigenvalues exact.*

α	$cond(A)$
10^{-1}	$8.4 * 10^3$
10^{-2}	$3.4 * 10^4$
10^{-3}	$2.8 * 10^5$
10^{-4}	$2.7 * 10^6$
10^{-5}	$2.7 * 10^7$

Table 10.2: Dependence of the condition number $cond(A)$ on α. The domain Ω is shown in Fig. 10.8 with $\beta = 1$, $a = 4h$ and 65^2 dofs.

dofs	$cond(A)$
9^2	$6.8 * 10^4$
17^2	$5.1 * 10^5$
33^2	$6.1 * 10^6$
65^2	$2.7 * 10^7$

Table 10.3: Dependence of the condition number on the number of degrees of freedom. The domain Ω is shown in Fig. 10.8 with $\alpha = 10^{-5}$, $\beta = 1$, $a = 4h$.

Remark 10.2.2 *The disadvantage of the CG method is its slow convergence because of a possibly large the condition number of A.*

The next theorem describes the convergence speed of the CG method.

Theorem 10.2.1 *Let Φ be a symmetric iteration. Its matrix W of the third normal form (i.e., $W(\mathbf{u}^{m+1} - \mathbf{u}^m) = A\mathbf{u}^m - \mathbf{c}$) is assumed to satisfy*

$$\lambda W \leq A \leq \Lambda W \quad (\lambda > 0).$$

Then the iterates $\mathbf{u}^m$ of the CG method applied to Φ fulfil the energy norm estimate

$$\|e^m\|_A \leq \frac{2c^m}{1 + c^{2m}} \|e^0\|_A,$$

with $c := \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} = \frac{\sqrt{\lambda_{max}}-\sqrt{\lambda_{min}}}{\sqrt{\lambda_{max}}+\sqrt{\lambda_{min}}}$ *and* $e^m = \mathbf{u}^m - \mathbf{u}^*$, $\mathbf{u}^*$ *is the exact solution of the system.*

Proof: see [32] p.274.

PCG Algorithm

We introduce the following notation: p^m is the search direction, $r^m = \mathbf{c} - A\mathbf{u}^m$, $m = 0, 1, ..., n-1$, is the residual. We stop the CG iterations when $\|r^m\|_2 = \varepsilon_{cg}$. If the matrix A is symmetric ($A = A^T$) and positive definite, i.e., $(Av, v) > 0$, the CG method can be applied. Let W^{-1} be a preconditioning.

Remark 10.2.3 *It is possible that* AW^{-1} *is not symmetric and therefore we apply the CG method to the system* $W^{-1/2}AW^{-1/2}\mathbf{u}' = W^{-1/2}b$. *If* A *and* W^{-1} *are symmetric, then* $W^{-1/2}AW^{-1/2}$ *is also symmetric.*

1. Start: $\mathbf{u}^0 := 0$, $r^0 := \mathbf{c} - A\mathbf{u}^0$, $p^0 := W^{-1}r^0$, $\rho := \langle p^0, r^0 \rangle$,
2. Iteration for $m = 0, 1, ...$(as long as $m < n$ and $\|r^m\|_2 \leq \varepsilon_{cg}$):

$$a^m := Ap^m, \quad \lambda_{opt} := \rho_m / \langle a^m, p^m \rangle,$$

$$\mathbf{u}^{m+1} := \mathbf{u}^m + \lambda_{opt} p^m,$$

$$r^{m+1} := r^m - \lambda_{opt} a^m,$$

$$q^{m+1} := W^{-1}r^{m+1}, \quad \rho_{m+1} := \langle q^{m+1}, r^{m+1} \rangle,$$

$$p^{m+1} := q^{m+1} + \frac{\rho_{m+1}}{\rho_m} p^m.$$

Remark 10.2.4 *One step* $\mathbf{u}^m \to \mathbf{u}^{m+1}$ *of the CG method requires one multiplication* Ap^m *and, in addition, only simple vector operations and scalar products. The matrix* A, *the vectors* $\mathbf{u}^m$, r^m *and* p^m *have to be stored.*

An implementation of the PCG method for the model problem is available in HLIB [28].

Remark 10.2.5 *The number of iterations of the CG method without preconditioning is proportional to* $\sqrt{cond(A)}$ *and the number of iterations with preconditioning is proportional to* $\sqrt{cond(W^{-1}A)}$. *If the preconditioning is chosen successfully, then* $\sqrt{cond(W^{-1}A)} \ll \sqrt{cond(A)}$.

To demonstrate how the CG method solves the problem (10.1) with $f = 1$, $g = 0$ and $\alpha(x, y) = 1 + \frac{1}{2}sin(20x)sin(20y)$ on the model grid (Fig. 10.1) we offer Table 10.4. The stiffness matrix A was approximated by an $\mathcal{H}$-matrix with the adaptive rank arithmetic ($\varepsilon_a = 10^{-4}$). Table 10.4 shows the preparation time (in sec.), the computational time (in sec.), the residual and the number of iterations for different numbers of degrees of freedom.

dofs	Preparation time	Computational time	$\|A\tilde{\mathbf{u}}_{cg} - \mathbf{c}\|_2$	$\#iter$
33^2	0.2	1.9	$7.7 * 10^{-5}$	82
65^2	0.92	18.8	$9.3 * 10^{-5}$	178
129^2	50.23	165.0	$9.3 * 10^{-5}$	381

Table 10.4: Computational time, residual and number of iterations for different numbers of degrees of freedom.

10.3 Smooth Coefficients

Table 10.5 shows that the HDD method works very well on the model Poisson problem (frequency $\nu = 0$). The analytic solution $u = x^2 + y^2$ is given. The relative and absolute errors show exponential decay with increasing the rank k in the $\mathcal{H}$-matrix arithmetic.

Table 10.6 shows that for different numbers of degrees of freedom (129^2 and 257^2) the relative and absolute errors decrease when the rank k increases. In the rows $k = 7, 8$ the sum of the discretisation error ε_h and the quadrature error becomes comparable with the $\mathcal{H}$-matrix approximation error.

k	$\|\mathbf{u} - \tilde{\mathbf{u}}_k\|_2 \,/\, \|\mathbf{u}\|_2$	$\|\mathbf{u} - \tilde{\mathbf{u}}_k\|_\infty$
2	$7.2 * 10^{-1}$	$8.6 * 10^{-1}$
3	$4.8 * 10^{-2}$	$1.2 * 10^{-1}$
4	$3.3 * 10^{-3}$	$1.5 * 10^{-2}$
5	$3.6 * 10^{-4}$	$2.0 * 10^{-3}$
8	$4.1 * 10^{-7}$	$3.92 * 10^{-6}$
12	$1.2 * 10^{-10}$	$3.1 * 10^{-9}$

Table 10.5: Dependence of the relative and absolute errors on the rank k. 129^2 dofs, $\alpha = 1$, $f = 4$, $u = x^2 + y^2$.

k	$\frac{\|\mathbf{u}-\tilde{\mathbf{u}}_k\|_2}{\|\mathbf{u}\|_2}$	$\|\mathbf{u}-\tilde{\mathbf{u}}_k\|_\infty$	$\frac{\|\mathbf{u}-\tilde{\mathbf{u}}_k\|_2}{\|\mathbf{u}\|_2}$	$\|\mathbf{u}-\tilde{\mathbf{u}}_k\|_\infty$
1	1.6	1.34	2.65	1.61
2	$5.7 * 10^{-1}$	$6.6 * 10^{-1}$	1.04	$9.8 * 10^{-1}$
3	$2.8 * 10^{-2}$	6.1^{-2}	$2.4 * 10^{-1}$	3.3^{-1}
4	$1.9 * 10^{-3}$	$7.1 * 10^{-3}$	$1.2 * 10^{-2}$	$3.2 * 10^{-2}$
5	$2.7 * 10^{-4}$	$8.6 * 10^{-4}$	$9.3 * 10^{-4}$	$3.7 * 10^{-3}$
6	$5.4 * 10^{-5}$	$1.1 * 10^{-4}$	$1.6 * 10^{-4}$	$7.9 * 10^{-4}$
7	$1.17 * 10^{-5}$	$3.15 * 10^{-5}$	$4.12 * 10^{-5}$	$1.2 * 10^{-4}$
8	$8.2 * 10^{-6}$	$1.05 * 10^{-5}$	$1.6 * 10^{-5}$	$3.6 * 10^{-5}$

Table 10.6: Dependence of the relative and absolute errors on the rank k for 129^2 (columns 2,3) and 257^2 (columns 4-5) dofs, $u = x^4 + y^2$, $f = 12x^2 + 1$, $\alpha = 1$.

HDD Time

Table 10.7 shows the computational time of the HDD method with $f_h \in V_H \subset V_h$, $\frac{H}{h} = 2$, for different numbers of degrees of freedom. The number of degrees of freedom is always increased by a factor 4 and the time increased by a factor not greater than 6 (note that the quadratic dependence results factor 16). After an appropriate optimization of the data structures and Algorithms 6.3.1 and 6.3.3 in the HDD method, it is possible to decrease the time-factor to achieve an almost linear dependence.

dofs in Ω	HDD time(sec)
33^2	0.19
65^2	0.96
129^2	5.6
257^2	36.1
513^2	218.4

Table 10.7: Dependence of the HDD computing time on the number of degrees of freedom. Performed on Kepler (see Tab. 10.1), $n_{min} = 48$, $k = 5$.

10.4 Oscillatory Coefficients

In this subsection we consider the model problem (10.1) with $f = 1$, $g = 0$, $\alpha(x, y) = 1 + Asin(\nu x)sin(\nu y)$. The aim is to show how the accuracy of the solution produced by HDD depends on the frequency ν and the amplitude A. Note that we research approximation properties of HDD and we do not care about the discretisation error. We suppose that the enough accurate discretisation is already done.

Example 10.4.1 *To demonstrate the oscillatory effect we plotted (Figure 10.2) the solution of the problem*

$$\begin{aligned} -\operatorname{div}(\alpha(\mathbf{x})\nabla u) &= 1 \quad \text{in } \Omega = (0,1)^2, \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned} \tag{10.2}$$

with $\alpha = \frac{1}{1.001+sin(150x)sin(150y)}$.

Very often the small details of the solution is out of interest and only the global behaviour of the solution is wanted!

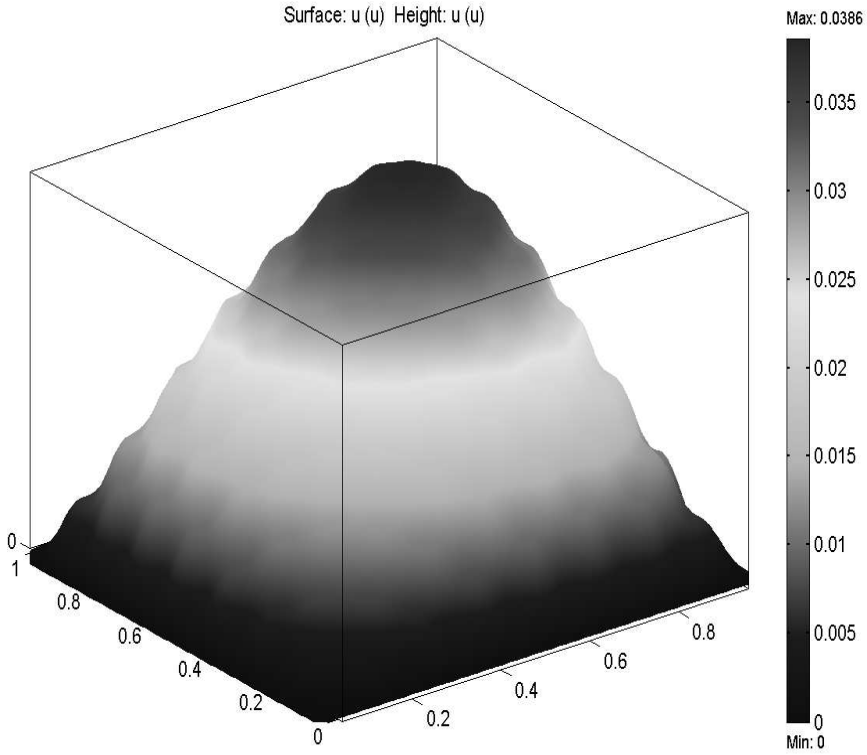


Figure 10.2: The solution of the problem (10.2).

Example 10.4.2 *Figures 10.3, 10.4 show the function* $\alpha(x,y) = 1+\frac{1}{2}sin(\nu x)sin(\nu y)$ *for* $\nu = 2$ *and* $\nu = 20$.

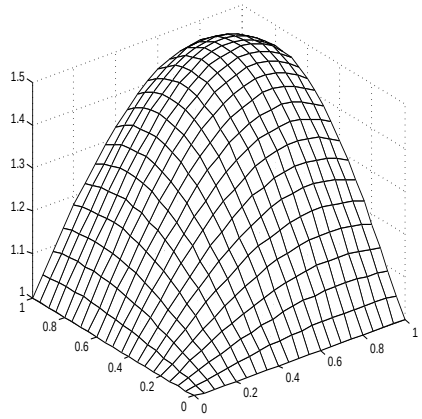


Figure 10.3: The coefficient function $\alpha(x,y) = 1 + \frac{1}{2}sin(2x)sin(2y)$.

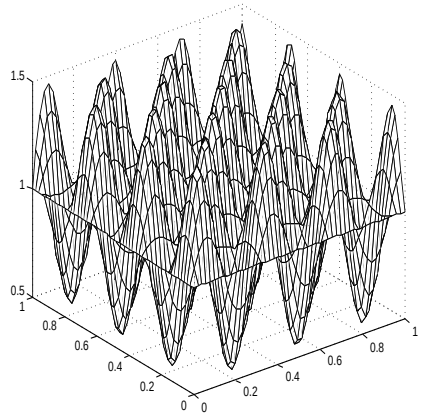


Figure 10.4: The coefficient function $\alpha(x,y) = 1 + \frac{1}{2}sin(20x)sin(20y)$.

Since $sin(\nu x)$ has period $\frac{2\pi}{\nu}$ (see Fig. 10.5), the interval $[0, \frac{2\pi}{\nu}]$ should contain at least 5 points, i.e., $\frac{2\pi}{\nu} \geq 4h$, where $h = \frac{1}{(N-1)}$. Thus, for $\nu \leq \frac{2\pi(N-1)}{4}$ there are more than 5 points in the interval $[0, \frac{2\pi}{\nu}]$ (upper figure) and for $\nu > \frac{2\pi(N-1)}{4}$ less then 5 points (lower figure).

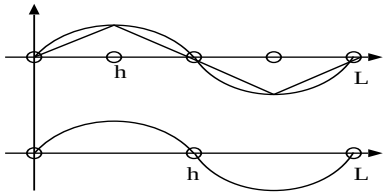


Figure 10.5: An P^1 approximation of $sin(x)$ on $[0, L]$ by 5 points (up) and by 3 points (bottom). In the last case, the approximating function is $\equiv 0$ and the discretisation error is large.

Table 10.8 shows how the HDD method solves the problems with oscillatory right-hand side. The analytic solution to the problem with oscillatory coefficients has been chosen to be non-oscillatory $u = x^2 + y^2$. The discrete analytic solution $\mathbf{u}$ is compared with the solution $\tilde{\mathbf{u}}_k$, obtained by the HDD method. One can see that for $k = 2, 3, 4, 5$ the both relative and absolute errors decrease. But for $k = 8, 12$ the quadrature errors (see (3.33)) and discretisation errors come into play and become to dominate.

k	$\|\mathbf{u} - \tilde{\mathbf{u}}_k\|_2 \,/\, \|\mathbf{u}\|_2$	$\|\mathbf{u} - \tilde{\mathbf{u}}_k\|_\infty$
2	$7.4 * 10^{-1}$	$8.8 * 10^{-1}$
3	$4.9 * 10^{-2}$	$1.3 * 10^{-1}$
4	$6.7 * 10^{-3}$	$3 * 10^{-2}$
5	$8.7 * 10^{-4}$	$4.9 * 10^{-3}$
8	$2.8 * 10^{-4}$	$6.9 * 10^{-4}$
12	$2.8 * 10^{-4}$	$6.9 * 10^{-4}$

Table 10.8: Dependence of the absolute and relative errors on the rank k. 129^2 dofs, $\alpha(x, y) = 1 + \frac{1}{2}sin(\nu x)sin(\nu y)$, $f = 4 + 2sin(\nu x)sin(\nu y) + x\nu cos(\nu x)sin(\nu y) + y\nu sin(\nu x)cos(\nu y)$, $\nu = 50$, $u = x^2 + y^2$.

Recall that the solution obtained by HDD with the maximal $\mathcal{H}$-matrix rank k is denoted by $\tilde{\mathbf{u}}_k$. The solution obtained by HDD with the maximal rank 40 be $\tilde{\mathbf{u}}_{40}$. Such large rank 40 results the almost exact matrix arithmetic. Tables 10.9, 10.10, 10.11 show how the accuracy of the HDD method depends on the accuracy of the $\mathcal{H}$-matrix approximation. Table 10.9 contains the relative and absolute errors of the solution to the problem 10.1 with $\alpha = 1 + \frac{1}{2}sin(50x)sin(50y)$ for 129^2 dofs (columns 2-3) and for 257^2 dofs (columns 4-5). One can see an exponential decay of both errors with increasing the $\mathcal{H}$-matrix rank k.

k	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_2 / \|\tilde{\mathbf{u}}_{40}\|_2$	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_\infty$	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_2 / \|\tilde{\mathbf{u}}_{40}\|_2$	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_\infty$
2	1.3	$4.8 * 10^{-2}$	6.1	$6.8 * 10^{-2}$
3	$5.0 * 10^{-2}$	$4.2 * 10^{-3}$	0.6	$3.2 * 10^{-2}$
4	$4.5 * 10^{-3}$	$7.9 * 10^{-4}$	$1.7 * 10^{-2}$	$2.4 * 10^{-3}$
6	$1.8 * 10^{-4}$	$1.9 * 10^{-5}$	$2.2 * 10^{-3}$	$3.1 * 10^{-4}$
7	$7.3 * 10^{-5}$	$7.76 * 10^{-6}$	$5.7 * 10^{-4}$	$5.6 * 10^{-5}$
8	$1.6 * 10^{-5}$	$1.8 * 10^{-6}$	$1.5 * 10^{-4}$	$1.6 * 10^{-5}$
9	$4.9 * 10^{-6}$	$5.3 * 10^{-7}$	$5.8 * 10^{-5}$	$5.5 * 10^{-6}$
10	$1.36 * 10^{-6}$	$2.0 * 10^{-7}$	$6 * 10^{-6}$	$6.5 * 10^{-7}$
12	10^{-7}	$1.6 * 10^{-8}$	$7.1 * 10^{-7}$	$8.6 * 10^{-8}$
14	$6.9 * 10^{-9}$	$1.2 * 10^{-9}$	$4.8 * 10^{-8}$	$8.2 * 10^{-9}$

Table 10.9: Dependence of the absolute and relative errors on the rank k. $f = 1$, $\alpha(x, y) = 1 + \frac{1}{2}sin(50x)sin(50y)$. (**2-3 columns**) 129^2 dofs, $\|\tilde{\mathbf{u}}_{40}\|_2 = 5.4$, (**4-5 columns**) 257^2 dofs, $\|\tilde{\mathbf{u}}_{40}\|_2 = 11.0$.

Tables 10.10, 10.11 demonstrate an exponential decay of both relative and absolute errors for the problem (10.1) with the oscillatory coefficient $\alpha(x, y) = 1 + \frac{1}{2}sin(10x)sin(10y)$. The experiments in Table 10.10 are done for 33^2, 65^2 degrees of freedom and in Table 10.11 for 129^2, 257^2 degrees of freedom.

k	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_2 / \|\tilde{\mathbf{u}}_{40}\|_2$	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_\infty$	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_2 / \|\tilde{\mathbf{u}}_{40}\|_2$	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_\infty$
2	$1.88 * 10^{-2}$	$1.95 * 10^{-3}$	$1.7 * 10^{-1}$	$1.37 * 10^{-2}$
3	$6.4 * 10^{-3}$	$6.6 * 10^{-4}$	$8.6 * 10^{-3}$	$7.4 * 10^{-4}$
4	$1.74 * 10^{-3}$	$1.9 * 10^{-4}$	$3.1 * 10^{-3}$	$2.9 * 10^{-4}$
5	$1.6 * 10^{-4}$	$2.2 * 10^{-5}$	$4.8 * 10^{-4}$	$5.4 * 10^{-5}$
6	$1.14 * 10^{-5}$	$2.27 * 10^{-6}$	$6.73 * 10^{-5}$	$8.9 * 10^{-6}$
7	$2.43 * 10^{-6}$	$4.9 * 10^{-7}$	$1.8 * 10^{-5}$	$2.54 * 10^{-6}$
8	$2.64 * 10^{-7}$	$4.6 * 10^{-8}$	$4.34 * 10^{-6}$	$6.9 * 10^{-7}$
9	$7.55 * 10^{-10}$	$1.65 * 10^{-10}$	$1.1 * 10^{-6}$	$1.9 * 10^{-7}$

Table 10.10: Dependence of the absolute and relative errors on the rank k. $f = 1$, $\alpha(x, y) = 1 + \frac{1}{2}sin(10x)sin(10y)$. (**2-3 columns**) 33^2 dofs, $\|\tilde{\mathbf{u}}_{40}\|_2 = 1.36$; (**4-5 columns**) 65^2 dofs, $\|\tilde{\mathbf{u}}_{40}\|_2 = 2.74$.

k	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_2 / \|\tilde{\mathbf{u}}_{40}\|_2$	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_\infty$	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_2 / \|\tilde{\mathbf{u}}_{40}\|_2$	$\|\tilde{\mathbf{u}}_{40} - \tilde{\mathbf{u}}_k\|_\infty$
3	$2.5 * 10^{-2}$	$2.3 * 10^{-3}$	$2.7 * 10^{-1}$	$2 * 10^{-2}$
4	$5.1 * 10^{-3}$	$4.3 * 10^{-4}$	$2.1 * 10^{-2}$	$2 * 10^{-3}$
5	$7.3 * 10^{-4}$	$6.8 * 10^{-5}$	$1.2 * 10^{-3}$	$1.3 * 10^{-4}$
6	$1.65 * 10^{-4}$	$1.76 * 10^{-5}$	$2.9 * 10^{-4}$	$2.6 * 10^{-5}$
7	$6.3 * 10^{-5}$	10^{-6}	$1.1 * 10^{-4}$	$1.1 * 10^{-5}$
8	$1.94 * 10^{-5}$	$2.3 * 10^{-6}$	$4.6 * 10^{-5}$	$4.5 * 10^{-6}$
9	$4.67 * 10^{-6}$	$6.2 * 10^{-7}$	$1.2 * 10^{-5}$	$1.4 * 10^{-6}$
10	$1.0 * 10^{-6}$	$1.6 * 10^{-7}$	$3.3 * 10^{-6}$	$4.1 * 10^{-7}$
12	$1.0 * 10^{-7}$	$1.7 * 10^{-8}$	$4.3 * 10^{-7}$	$5.5 * 10^{-8}$
14	$1.1 * 10^{-8}$	$2.1 * 10^{-9}$	$5.9 * 10^{-8}$	$9.5 * 10^{-9}$
16	$9.0 * 10^{-10}$	$2.13 * 10^{-10}$	$7.4 * 10^{-9}$	$1.3 * 10^{-9}$

Table 10.11: Dependence of the absolute and relative errors on the rank k. $f = 1$, $\alpha(x, y) = 1 + \frac{1}{2} sin(10x) sin(10y)$. (**2-3 columns**) 129^2 dofs, $\|\tilde{\mathbf{u}}_{40}\|_2 = 5.5$; (**4-5 columns**) 257^2 dofs, $\|\tilde{\mathbf{u}}_{40}\|_2 = 11.0$.

In Table 10.12 we compare the solution of the initial problem (10.1), obtained by the HDD method with the solution obtained by the PCG method. The maximal number of iterations for the PCG method is 600, the admissible value of the residual $\|A\tilde{\mathbf{u}}_{cg} - \mathbf{c}\|_2 = 10^{-6}$. Thus, we can assume that the PCG method gives an 'exact' FE solution. The errors in columns 2 and 3 increase because the $\mathcal{H}$-matrix approximation error $\varepsilon_{\mathcal{H}}$ becomes larger. As a preconditioner we use the $\mathcal{H}$-Cholesky factorization.

dofs	$\|\tilde{\mathbf{u}}_{cg} - \tilde{\mathbf{u}}_\varepsilon\|_\infty$	$\frac{\|\tilde{\mathbf{u}}_{cg} - \tilde{\mathbf{u}}_\varepsilon\|_2}{\|\mathbf{u}_\varepsilon\|_2}$	$\|A\tilde{\mathbf{u}}_{cg} - \mathbf{c}\|_2$	time$_{HDD}$, (sec)
17^2	$4.42 * 10^{-10}$	$4.34 * 10^{-9}$	$9.1 * 10^{-7}$	0.06
33^2	$8.56 * 10^{-7}$	$5.1 * 10^{-6}$	$7.2 * 10^{-7}$	0.27
65^2	$4.21 * 10^{-6}$	$3.3 * 10^{-5}$	$9.6 * 10^{-7}$	1.6
129^2	$1.32 * 10^{-5}$	$1.3 * 10^{-4}$	$9.7 * 10^{-7}$	10.0

Table 10.12: Comparison of the HDD solution $\tilde{\mathbf{u}}_\varepsilon$ with the PCG solution $\tilde{\mathbf{u}}_{cg}$, $\alpha(x, y) = 1 + \frac{1}{2} sin(50x) sin(50y)$, $\varepsilon_a = 10^{-6}$.

In Table 10.13 we consider the problem (10.1) with

$$\alpha(x, y) = \frac{2 + P \sin(2\pi x/\varepsilon)}{2 + P \sin(2\pi y/\varepsilon)} + \frac{2 + P \sin(2\pi y/\varepsilon)}{2 + P \sin(2\pi x/\varepsilon)},$$

$f = 1$, $P = 1.8$, $\varepsilon = 1/512$ and $g = 0$. The domain Ω and its triangulation are shown in Proposition 7.1.2. We assume that the solution, obtained by the PCG-method with the $\mathcal{H}$-Cholesky preconditioning is the exact solution (of course, up to the discretisation error). The residual for the PCG-method is $\|A\tilde{u}_{cg} - \mathbf{c}\|_2 = 10^{-6}$ and the number of iterations is shown in brackets. The $\mathcal{H}$-matrix computations were

done with the adaptive rank arithmetic $\varepsilon_a = 10^{-5}$ (see Def. 5.9.3). One can see that HDD achieves the same accuracy as PCG with a similar computational time. Note that both methods does not take into account the discretisation error.

dofs	$\|\tilde{\mathbf{u}}_{cg} - \tilde{\mathbf{u}}_\varepsilon\|_2$	$\|\tilde{\mathbf{u}}_{cg} - \tilde{\mathbf{u}}_\varepsilon\|_\infty$	PCG sec.(iter)	HDD sec.
33^2	$5.7 * 10^{-4}$	$4.5 * 10^{-5}$	0.17(4)	0.39
65^2	$2.8 * 10^{-4}$	$1.1 * 10^{-5}$	1.18(4)	1.84
129^2	$1.4 * 10^{-4}$	$2.86 * 10^{-6}$	9.6(6)	10.5

Table 10.13: Dependence of the absolute error on the number of degrees of freedom. $\varepsilon_a = 10^{-5}$.

10.5 Comparison of HDD With $\mathcal{H}$-Matrix Inverse and Inverse by Cholesky Decomposition

In this section we compare the computational time and memory requirement of HDD with the times and memory requirements of the $\mathcal{H}$-Matrix inverse and the $\mathcal{H}$-Cholesky decomposition.
Table 10.14 demonstrates the dependence of memory requirements and the computational times on the parameter ε_a (see the adaptive rank arithmetic in Def. 5.9.3). One can see that the computational time and storage requirement of the $\mathcal{H}$-Cholesky factorisation are the best. The HDD method shows the sligtly larger time than the $\mathcal{H}$-Cholesky factorisation and a much better time as the direct $\mathcal{H}$-matrix inverse. HDD requires more memory than the $\mathcal{H}$-Cholesky factorisation and much less than the direct $\mathcal{H}$-matrix inverse. The memory requirements for HDD can be decreased after optimization of the Algorithms 6.3.1 and 6.3.3. Note that HDD computes the solution operators, but after the $\mathcal{H}$-Cholesky factorization one still needs to solve two systems of linear equations $L\mathbf{v} = \mathbf{c}$ and $L^T\mathbf{u} = \mathbf{v}$.

ε_a	$\mathcal{H}$-Cholesky time;size	HDD-time;size	time($A^{-\mathcal{H}}$);size
10^{-3}	2.1;(13.3)	9.2;(19.7)	21.4;(51.0)
10^{-4}	2.6;(14.7)	9.8;(20.1)	29.6;(64.0)
10^{-5}	3.0;(16.0)	10.6;(20.4)	37.3;(75.2)
10^{-6}	3.4;(17.2)	11.6;(20.6)	47.4;(87.4)

Table 10.14: Comparison of the $\mathcal{H}$-Cholesky factorisation, HDD and the $\mathcal{H}$-matrix inverse. Dependence of time (in sec.) and memory requirements (in MB) on ε_a, 129^2 dofs.

Table 10.15 shows the dependence of the computational time for the $\mathcal{H}$-Cholesky factorisation on the number of degrees of freedom. The complexity of the $\mathcal{H}$-matrix arithmetic (see Table 5.3) depends on the factor k^2 and it is why we do not see almost linear factor in time. It is shown ([46], [47]) that for a smaller $\mathcal{H}$-matrix rank k an almost linear complexity can be achieved. Note that it is not enough memory for computing $\mathcal{H}$-Cholesky factorization with 513^2 dofs and $k = 8$. The error $\left\|(LL^T)^{-1}A - I\right\|_2$ grows up because the $\mathcal{H}$-matrix approximation error grows up.

dofs	$\left\|(LL^T)^{-1}A - I\right\|_2$	$\|\mathbf{u} - \tilde{\mathbf{u}}_L\|_2 \,/\, \|\tilde{\mathbf{u}}_L\|_2$	t, sec
33^2	$2.03 * 10^{-14}$	$1.5 * 10^{-15}$	0.08
65^2	$5.6 * 10^{-11}$	$6.1 * 10^{-13}$	0.8
129^2	$9.0 * 10^{-10}$	$1.1 * 10^{-11}$	6.68
257^2	$6.7 * 10^{-9}$	$9.9 * 10^{-11}$	75.0
513^2	n.e.m.	n.e.m.	not enough memory

Table 10.15: The computational time and accuracy of the solution $\tilde{\mathbf{u}}_L$, obtained by the $\mathcal{H}$-Cholesky decomposition, $u = (x^2 - 1)(y^2 - 1)$, $k = 8$.

Table 10.16 compares the computational times of the HDD method and the PCG method with $\mathcal{H}$-Cholesky preconditioner. Column 3 contains the measurements of times for: (a) computing the stiffness matrix A in the data-sparse format; (b) computing the $\mathcal{H}$-Cholesky decomposition of A (used as a preconditioner); (c) PCG iterations.
Note that for 513^2 dofs there is not enough memory to compute the stiffness matrix A and perform its $\mathcal{H}$-Cholesky factorization. The advantage of the HDD method is that it does not require agglomeration of the whole stiffness matrix. The memory is dynamically allocated and deallocated.

dofs	HDD	PCG
33^2	0.19	0.1=0.03+0.04+0.02
65^2	0.96	0.6=0.2+0.26+0.1
129^2	5.6	5=2.6+1.8+0.6
257^2	36.1	53=38.0+11.4+3.4
513^2	218	n.e.m.

Table 10.16: Comparison of times for the skin problem with $\alpha = 10^{-5}$, $\varepsilon_a = 10^{-8}$, $\varepsilon_{cg} = 10^{-8}$, $\frac{H}{h} = 2$. Performed on Kepler from Table 10.1.

Table 10.17 shows how the absolute and relative errors depends on the frequency ν. One can see the errors of the same order, i.e., the HDD method is stable with respect to the frequency ν. Here $\mathbf{u}_{40}$ is the solution computed by HDD in the class of $\mathcal{H}$-matrices with the maximal rank 40. Since the exact solution is unknown, we use

$\mathbf{u}_{40}$ as a good approximation of the exact solution (up to the discretisation error). Here it is quite appropriate to quote the work [11] about existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. In

ν	$\|\mathbf{u}_{40} - \tilde{\mathbf{u}}_k\|_2 \,/\, \|\mathbf{u}_{40}\|_2$	$\|\mathbf{u}_{40} - \tilde{\mathbf{u}}_k\|_\infty$
10	$1.65 * 10^{-4}$	$1.76 * 10^{-5}$
50	$1.8 * 10^{-4}$	$1.9 * 10^{-5}$

Table 10.17: Dependence of the relative and absolute errors on the frequency ν, 257^2 dofs, $f = 1$, $\alpha(x, y) = 1 + \frac{1}{2}sin(\nu x)sin(\nu y)$, $\varepsilon_a = 10^{-6}$.

Table 10.18 we compare the HDD method with the PCG method. Here $\tilde{\mathbf{u}}_\varepsilon$ is the HDD solution, $\tilde{\mathbf{u}}_{cg}$ the solution obtained by the PCG method with the $\mathcal{H}$-Cholesky preconditioner. The fourth and fifth columns present the computational times. The HDD time is comparable with the PCG time, but for 513^2 dofs PCG requires too much memory.

dofs	$\|\tilde{\mathbf{u}}_{cg} - \tilde{\mathbf{u}}_\varepsilon\|_2$	$\|\tilde{\mathbf{u}}_{cg} - \tilde{\mathbf{u}}_\varepsilon\|_\infty$	PCG with LL^T sec.(iter)	HDD sec.
33^2	$4.16 * 10^{-7}$	$6.68 * 10^{-8}$	0.22(2)	0.35
65^2	$2.28 * 10^{-5}$	$1.42 * 10^{-6}$	1.66(2)	2.5
129^2	$2.38 * 10^{-4}$	$9.35 * 10^{-6}$	17(2)	13
257^2	$2.35 * 10^{-3}$	$2.85 * 10^{-5}$	63(11)	60.6
513^2	n.e.m.	n.e.m.	not enough memory	270.3

Table 10.18: Dependence of the absolute errors on the number of dofs, $f = 1$, $\alpha(x, y) = 1/(1.0001 + sin(500x)sin(500y))$. All computations were performed on Notebook (see Table 10.1) with the adaptive rank arithmetic (see Def. 5.9.3), $\varepsilon_a = 10^{-5}$, $\frac{H}{h} = 2$.

Remark 10.5.1 *Note that the computational time of the $\mathcal{H}$-Cholesky factorisation depends on the rank k used in the $\mathcal{H}$-matrix arithmetic (see [47], [46]). If k becomes smaller than the computational time and the accuracy decreases also.*

Remark 10.5.2 *The later experiments for discontinuous coefficients [43] show that it is better (in the sense of timing performances) to take a smaller rank k with a larger number of PCG iterations as a larger k with a smaller number of PCG iterations.*

10.6 Memory Requirements for Φ^g and Φ^f

Tables 10.19, 10.20 show the total storage requirements for all matrices Φ^g_ω and Φ^f_ω, $\omega \in T_{\mathcal{T}_h}$ in the case of one grid and two grids. In Table 10.19 we see an almost linear dependence of the storage requirements on the number of degrees of freedom. The columns $S(\Phi^g)$ and $S(\Phi^f)$ present memory requirements for all mappings Φ^g_ω and Φ^f_ω, $\omega \in T_{\mathcal{T}_h}$, respectively. We see a factor ≈ 4 in the column $S(\Phi^g)$ and a factor ≈ 5.6 in $S(\Phi^f)$. Here the factor 4 shows a linear dependence. The numbers in this table are in accordance with the theoretical estimates in Lemma 7.3.4. We do not see a linear factor because there is an additional log factor.

dofs	$S(\Phi^g)$, Kb	$S(\Phi^f)$, Kb	$\|\mathbf{u}-\tilde{\mathbf{u}}_k\|_2 / \|\mathbf{u}\|_2$	$\|\mathbf{u}-\tilde{\mathbf{u}}_k\|_\infty$
33^2	$2.45 * 10^2$	$4 * 10^2$	$3.3 * 10^{-5}$	$8.47 * 10^{-5}$
65^2	$1.1 * 10^3$	$2.4 * 10^3$	$5.75 * 10^{-5}$	$1.0 * 10^{-4}$
129^2	$5 * 10^3$	$1.4 * 10^4$	$7.4 * 10^{-5}$	$1.1 * 10^{-4}$
257^2	$2.1 * 10^4$	$7.86 * 10^4$	$8.3 * 10^{-5}$	$1.3 * 10^{-4}$

Table 10.19: Dependence of the total memory requirements for all Φ^g_ω and Φ^f_ω on the number of degrees of freedom, $k = 7$, $u = x^2 + y^2$.

Table 10.20 shows the storage requirements for all Φ^g_ω and Φ^f_ω in dependence on the compression factor $\frac{H}{h}$. The storage requirement for all Φ^g_ω stays the same (column $S(\Phi^g)$) since for all mappings Φ^g_ω we use only one scale with step size h. Lemmas 7.3.4 and 7.4.5 state that $S(\Phi) \leq C_1 k n_h \log^2 n_h$ for one grid and $S(\Phi^f) \leq C_2 k \sqrt{n_H n_h} \log^2 \sqrt{n_H n_h}$ for two grids.

All computations in Table 10.20 were performed with the adaptive rank arithmetic $\varepsilon_a = 10^{-8}$.

$\frac{H}{h}$	$S(\Phi^g)$, MB	$S(\Phi^f)$, MB	time, sec
1	$2.2 * 10^1$	$2.9 * 10^2$	218
2	$2.2 * 10^1$	$8.7 * 10^1$	83
4	$2.2 * 10^1$	$3.2 * 10^1$	41
8	$2.2 * 10^1$	$1.8 * 10^1$	32
16	$2.2 * 10^1$	$1.5 * 10^1$	26

Table 10.20: The computational time and the total storage requirements for all Φ^g_ω and Φ^f_ω. 257^2 dofs, $\varepsilon_a = 10^{-8}$, $\alpha(x,y) = 1 + \frac{1}{2} sin(50x) sin(50y)$, $f(x,y) = x(x-1) + y(y-1)$.

10.7 Approximation of Φ^g and Φ^f

To estimate the accuracy of the $\mathcal{H}$-matrix approximation to the inverse of the global stiffness matrix A we compute the error $\|A \cdot A^{-\mathcal{H}} - I\|_2$. But in the HDD method we do not have the global matrix A, we have a set of matrices Φ^g_ω and Φ^f_ω, $\omega \in T_{\mathcal{T}_h}$. The matrices Φ^g_ω (as well as Φ^f_ω) on the l-th level of $T_{\mathcal{T}_h}$ are equivalent. It is why we consider the accuracy of the $\mathcal{H}$-matrix approximation only for one of them. The mappings $\Phi^g := \Phi^g_{127}$ and $\Phi^f := \Phi^f_{127}$ are computed with the $\mathcal{H}$-matrix rank $k = 127$ (up to almost machine precision 10^{-16}). Tables 10.21, 10.22 show the maximal operator errors $\|\Phi^g - \Phi^g_{21}\|_2$ and $\|\Phi^f - \Phi^f_7\|_2$ on different levels of $T_{\mathcal{T}_h}$. Φ^g_{21} and Φ^f_7 are the $\mathcal{H}$-matrix approximations of Φ^g and Φ^f with the maximal ranks 21 and 7 correspondingly. The second column shows the size of the matrix and the third column shows the corresponding level of the hierarchical domain decomposition tree $T_{\mathcal{T}_h}$. The maximal error appears at 0-level (root). It can be explained by the fact that the chosen maximal rank is insufficient for larger matrices. Note that the errors in Tables 10.21, 10.22 do not depend on the right-hand side.

Tables 10.21, 10.22 show that it is a nice idea to choose an adaptive rank for each

$\|\Phi^g - \Phi^g_{21}\|_2$	size of Φ^g	level of $T_{\mathcal{T}_h}$
$4.8 * 10^{-1}$	255×1024	0
$5.5 * 10^{-3}$	127×512	1
$4.22 * 10^{-5}$	63×256	2
$4.55 * 10^{-8}$	31×128	3
$2.58 * 10^{-17}$	15×64	4

Table 10.21: The error $\|\Phi^g - \Phi^g_{21}\|_2$ on different levels of $T_{\mathcal{T}_h}$. Φ^g_{21} is an approximation of Φ^g by a rank-21 matrix. The matrix size is $|I(\gamma_\omega)| \times |I(\partial\omega)|$, $\alpha(x, y) = 1 + \frac{1}{2} sin(50x) sin(50y)$, 127^2 dofs.

$\|\Phi^f - \Phi^f_7\|_2$	size of Φ^f	level of $T_{\mathcal{T}_h}$
$2.1 * 10^{-7}$	255×257^2	0
$3.6 * 10^{-9}$	127×129^2	1
$1.04 * 10^{-9}$	63×65^2	2
$2.04 * 10^{-10}$	31×33^2	3
$1.47 * 10^{-20}$	15×17^2	4

Table 10.22: The error $\|\Phi^f - \Phi^f_7\|_2$ on different levels of $T_{\mathcal{T}_h}$, Φ^f_7 is an approximation of Φ^f by rank-7 matrix. The matrix size is $|I(\gamma_\omega)| \times |I(\omega)|$, $\alpha(x, y) = 1 + \frac{1}{2} sin(50x) sin(50y)$, 127^2 dofs.

level of $T_{\mathcal{T}_h}$. For example, if $T_{\mathcal{T}_h}$ has L levels, than one may take $k_0 > k_1 > k_2 ... > k_{L-1}$, where k_i is the $\mathcal{H}$-matrix rank on the i-th level. The adaptive rank arithmetic (see Def. 5.9.3) realizes this idea.

10.8 Jumping Coefficients

In this section we consider the class of problems with jumping coefficients. Such problems appear in the material sciences (electrical fields through materials with different conductivities), in medicine (the so-called skin problem) etc. An simple example is shown in Fig. 10.6. This domain Ω has areas with jumping coefficients: $\alpha = 10$ and $\beta = 10^{-2}$.

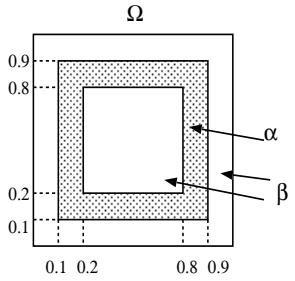


Figure 10.6: Domain $\Omega = (0,1)^2$ with jumping coefficients α and β.

Table 10.23 shows that the parameter ε_a (for the adaptive rank arithmetic) has to be smaller for problems with jumping coefficients than the corresponding parameter (denote by $\varepsilon_a^{\text{model}}$) for the model problem. A priori one can take

$$\varepsilon_a := \max_{\omega_i,\omega_j \in \mathcal{T}_h} \frac{\alpha_{\omega_i}}{\alpha_{\omega_j}} * \varepsilon_a^{\text{model}}.$$

Here $\max_{\omega_i,\omega_j \in \mathcal{T}_h}$ describes the maximal jump between two finite elements ω_i and ω_j (leaves of the tree $T_{\mathcal{T}_h}$).

ε_a	$\|AA^{-\mathcal{H}} - I\|_2$	time (sec.)
10^{-6}	1.2	63.7
10^{-8}	$3.0 * 10^{-1}$	87.1
10^{-10}	$3.4 * 10^{-4}$	111.6
10^{-12}	$3.7 * 10^{-6}$	156.8

Table 10.23: Dependence of the $\mathcal{H}$-matrix approximation error $\|AA^{-\mathcal{H}} - I\|_2$ on ε_a for the domain as in Fig. 10.6 with coefficients $\alpha = 10$ and $\beta = 0.01$, 129^2 dofs.

Table 10.23 shows the total time for the $\mathcal{H}$-matrix approximation (denoted by $A^{-\mathcal{H}}$) to the inverse A^{-1}. Here the time of building the block cluster tree is negligible small in comparison with the time of computing $A^{-\mathcal{H}}$. We see also that the accuracy of the $\mathcal{H}$-matrix approximation to A^{-1} increases with decreasing the parameter ε_a.

Table 10.24 compares the HDD method with the PCG method for different ε_a. The domain Ω is shown in Fig. 10.6. Such domains are typical, for instance, in electrostatics. The jumping coefficients α and β model electroconductivity in different materials. This table compares also the computational times. We see that the HDD time is larger. The reason is that HDD computes the solution operators, but PCG only the solution.

ε_a	$\|A\tilde{\mathbf{u}}_{cg} - \mathbf{c}\|_2$	PCG-time (sec)	HDD-time	$\frac{\|\tilde{\mathbf{u}}_{cg} - \tilde{\mathbf{u}}_\varepsilon\|_2}{\|\tilde{\mathbf{u}}_\varepsilon\|_2}$	$\|\tilde{\mathbf{u}}_{cg} - \tilde{\mathbf{u}}_\varepsilon\|_\infty$
10^{-4}	$2 * 10^{-4}$	5.3	8.9	$6.7 * 10^{-1}$	1.4
10^{-6}	$4.8 * 10^{-7}$	5.0	10.1	$1.8 * 10^{-4}$	$9.5 * 10^{-4}$
10^{-8}	$1.4 * 10^{-8}$	5.7	11.5	$1.1 * 10^{-6}$	$1.48 * 10^{-5}$
10^{-10}	$1.45 * 10^{-8}$	6.7	12.3	$5.3 * 10^{-7}$	10^{-5}
10^{-12}	$1.2 * 10^{-8}$	7.4	13.5	$5.2 * 10^{-7}$	10^{-5}

Table 10.24: Dependence of the relative and the absolute errors on ε_a for the model problem on the domain as in Fig. 10.6 with coefficients $\alpha = 10$, $\beta = 0.01$ and 129^2 dofs.

10.9 Skin Problem

The problem in Fig. 10.7 models a more difficult problem (so-called the *skin problem*). In this problem an ointment penetrates through the skin. The diffusion process is very slow inside the cells and much faster in the channels in between (so-called the *lipid layer*). We choose a rectangular quasi-uniform grid on $\Omega = (0,1)^2$ which is compatible with the lipid layer. The condition number $cond(A)$ in problems with jumping coefficients is proportional to $h^{-2} \max\limits_{\omega_i,\omega_j \in \mathcal{T}_h} \dfrac{\alpha(\omega_i)}{\alpha(\omega_j)}$, where $\alpha(\omega_i)$ is the jumping coefficient in ω_i, h the grid step size and ω_i, $\omega_j \in \mathcal{T}_h$.

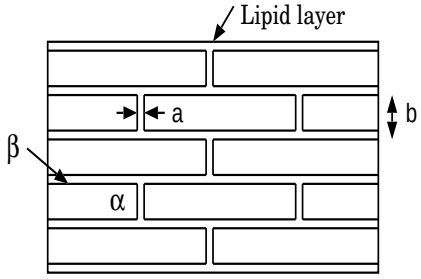


Figure 10.7: Model of a skin fragment $\Omega = (0,1)^2$. The coefficient of the penetration inside the cells is very small (α), but is large in between (β).

We model this difficult geometry by a simpler one as it shown in Fig. 10.8.

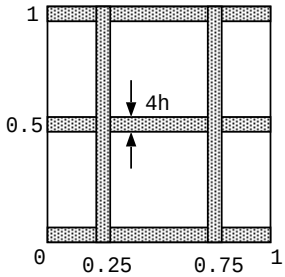


Figure 10.8: Model domain $\Omega = (0,1)^2$. The coefficient of penetration inside the cells is very small (α), but is large in between ($\beta = 1$).

Table 10.25 shows the accuracy of the HDD method for different jumping coefficients α (Fig. 10.8). We compare the solution, obtained by the HDD method, with the solution, obtained by PCG. For a small α (e.g., $\alpha = 10^{-5}$) the adaptive rank parameter $\varepsilon_a = 10^{-6}$ (Def. 5.9.3) is not good enough. For highly jumping coefficient one should choose very small parameter ε_a.

α	$\|A\tilde{\mathbf{u}}_{cg} - \mathbf{c}\|_2$	$\frac{\|\tilde{\mathbf{u}}_{cg} - \tilde{\mathbf{u}}_\varepsilon\|_2}{\|\tilde{\mathbf{u}}_\varepsilon\|_2}$	$\|\tilde{\mathbf{u}}_{cg} - \tilde{\mathbf{u}}_\varepsilon\|_\infty$
10^{-1}	$1.4 * 10^{-10}$	$8.0 * 10^{-6}$	$4.5 * 10^{-6}$
10^{-2}	$3.1 * 10^{-10}$	$1.6 * 10^{-5}$	$6.2 * 10^{-5}$
10^{-3}	$7.7 * 10^{-8}$	$5.7 * 10^{-5}$	$2.8 * 10^{-3}$
10^{-4}	$1.3 * 10^{-9}$	$7.0 * 10^{-3}$	1.5
10^{-5}	$8.9 * 10^{-9}$	$7.7 * 10^{-1}$	$8.8 * 10^{2}$

Table 10.25: Dependence of the absolute and relative errors on the coefficient α. 129^2 dofs, $\varepsilon_a = 10^{-6}$, domain as in Fig. 10.8 with $\beta = 1$ and thickness $a = 4h$.

Table 10.26 shows that the larger the jump is ($\sim$ the smaller α is) the larger are the relative and the absolute errors in the infinity, energy and spectral norms. We recall that A is the stiffness matrix for the whole domain Ω, $\tilde{\mathbf{u}}_{cg}$ is the solution computed by the PCG method and $\tilde{\mathbf{u}}_{\varepsilon}$ is the solution computed by the HDD method.

α	$\frac{\|\tilde{\mathbf{u}}_{cg}-\tilde{\mathbf{u}}_{\varepsilon}\|_2}{\|\tilde{\mathbf{u}}_{cg}\|_2}$	$\|\tilde{\mathbf{u}}_{cg}-\tilde{\mathbf{u}}_{\varepsilon}\|_{\infty}$	$\|\tilde{\mathbf{u}}_{cg}-\tilde{\mathbf{u}}_{\varepsilon}\|_{A}$	$\|A\tilde{\mathbf{u}}_{cg}-A\tilde{\mathbf{u}}_{\varepsilon}\|_2$	$\|A\|_2$
1.0	$6.6*10^{-9}$	$7.1*10^{-10}$	$2.3*10^{-7}$	$2.8*10^{-5}$	$1.27*10^{5}$
10^{-1}	$2.0*10^{-8}$	$1.4*10^{-8}$	$2.0*10^{-6}$	$1.1*10^{-4}$	$1.22*10^{5}$
10^{-2}	$6.6*10^{-8}$	$2.6*10^{-7}$	$1.7*10^{-5}$	$6.9*10^{-4}$	$1.22*10^{5}$
10^{-3}	$7.4*10^{-7}$	$1.8*10^{-5}$	$4.2*10^{-4}$	$8.8*10^{-3}$	$1.22*10^{5}$
10^{-4}	$4.2*10^{-6}$	$1.8*10^{-3}$	$1.4*10^{-2}$	$7.5*10^{-2}$	$1.22*10^{5}$
10^{-5}	$7.0*10^{-5}$	$2.3*10^{-1}$	$9.0*10^{-1}$	1.0	$1.22*10^{5}$

Table 10.26: Dependence of the absolute and relative errors on the jumping coefficient α. Ω as in Fig. 10.8, 129^2 dofs, thickness $a = 4h$, $\beta = 1$. $\varepsilon_a = 10^{-8}$, residual $\|A\tilde{\mathbf{u}}_{cg} - \mathbf{c}\|_2 = 10^{-10}$.

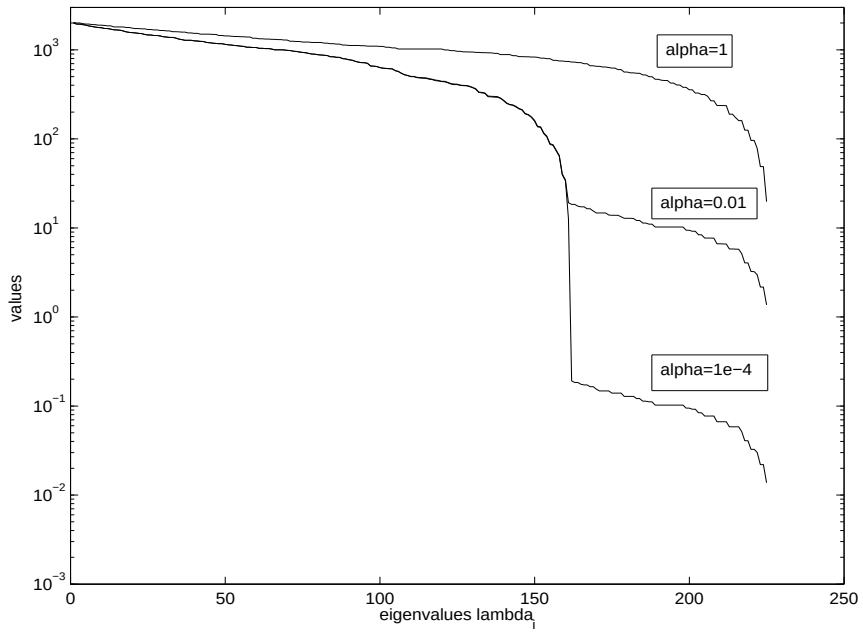
The accuracy of the solution $\tilde{\mathbf{u}}_{\varepsilon}$ depends on the accuracy of the $\mathcal{H}$-matrix approximation. We use the adaptive rank arithmetic, i.e., we choose the rank for each submatrix as follows $k = \min\{i : \sigma_i \le \varepsilon_a \sigma_1\}$, where $\sigma_1 \le \sigma_2 \le ... \le \sigma_k \le ...$ are the singular values.
Table 10.27 shows the dependence of the absolute and relative errors on the parameter ε_a (for the adaptive rank arithmetic). One can see that the accuracy of HDD can be improved by decreasing ε_a.

The comparison of the solution $\tilde{\mathbf{u}}_L$ computed by the direct $\mathcal{H}$-Cholesky with $\tilde{\mathbf{u}}_{cg}$ (computed by PCG) is given in the last column. We assume that PCG after a large number of iteration steps produces the 'exact' FE solution $\tilde{\mathbf{u}}_{cg}$.

ε_a	$\frac{\|\tilde{\mathbf{u}}_{cg}-\tilde{\mathbf{u}}_{\varepsilon}\|_2}{\|\tilde{\mathbf{u}}_{cg}\|_2}$	$\|\tilde{\mathbf{u}}_{cg}-\tilde{\mathbf{u}}_{\varepsilon}\|_{\infty}$	$\|\tilde{\mathbf{u}}_{cg}-\tilde{\mathbf{u}}_{\varepsilon}\|_{A}$	$\|A\tilde{\mathbf{u}}_{cg}-A\tilde{\mathbf{u}}_{\varepsilon}\|_2$	$\frac{\|\tilde{\mathbf{u}}_{cg}-\mathbf{u}_{L}\|_2}{\|\tilde{\mathbf{u}}_{cg}\|_2}$
10^{-6}	$4.4*10^{-1}$	$6.67*10^{2}$	$1.1*10^{3}$	$1.8*10^{2}$	$4.7*10^{-4}$
10^{-8}	$7.27*10^{-5}$	$2.3*10^{-1}$	$9.0*10^{-1}$	1.0	$6*10^{-7}$
10^{-10}	$5.1*10^{-7}$	$1.0*10^{-3}$	$3.0*10^{-3}$	$6.1*10^{-3}$	$1.1*10^{-8}$
10^{-12}	$3.9*10^{-9}$	$1.2*10^{-5}$	$2.9*10^{-5}$	$3.8*10^{-5}$	$1*10^{-11}$
10^{-14}	$1.2*10^{-11}$	$6.6*10^{-7}$	$1.2*10^{-7}$	$3.7*10^{-7}$	$1.5*10^{-13}$
10^{-16}	$1.6*10^{-12}$	$1.1*10^{-8}$	$1.7*10^{-8}$	$5.3*10^{-7}$	$2.1*10^{-14}$

Table 10.27: Dependence of the absolute and relative errors on ε_a. Ω as in Fig. 10.8, 129^2 dofs, thickness $a = 4h$, $\alpha = 10^{-5}$, $\beta = 1$. The residual is $\|A\tilde{\mathbf{u}}_{cg} - \mathbf{c}\|_2 = 10^{-10}$, $\|A\|_2 = 1.22*10^{5}$.

In the figure below one can see the spectra of the operator $\mathrm{div}(\alpha(\mathbf{x})\nabla)$ for $\beta = 1$, $\alpha = 1$, $\alpha = 10^{-2}$ and $\alpha = 10^{-4}$ (three curves in log-scale). The number of degrees of freedom is 33^2. This Figure helps us to understand why the condition number of the stiffness matrix for the skin problem (Figure 10.8) is large. All three maximal eigenvalues are almost the same and the minimum eigenvalues are differ by a factor proportional to the jump.



Decay of singular values of the stiffness matrix A for $\alpha = 1$, $\alpha = 10^{-2}$ and $\alpha = 10^{-4}$.

10.10 Problems With Many Right-Hand Sides

In the following experiment we show that the HDD method is well suited to problems with many right-hand sides. We consider the following problem

$$\begin{aligned} -\operatorname{div}(\alpha\nabla u) &= f^{(i)} \quad \text{in } \Omega, \\ u &= g^{(i)} \quad \text{on } \partial\Omega, \end{aligned} \tag{10.3}$$

where $i = 1, ..., i_{max}$ and Ω shown in Fig. 10.8. The number of degrees of freedom is 257^2. There are two grids with step sizes h and $H := 2h$. The right-hand side $f_H^{(i)} \in V_H$ (see Sec. 4.3.6), i.e., $f_H^{(i)} \in \mathbb{R}^{129^2}$. The solution $u_h^{(i)} \in V_h$ and $u_h^{(i)} \in \mathbb{R}^{257^2}$. The HDD method computes all the mappings Φ_ω^f and Φ_ω^g, $\omega \in T_{\mathcal{T}_h}$, once and then applies them in order to compute the solution $u_h^{(i)}$.
As an alternative approach we choose the PCG method with the $\mathcal{H}$-Cholesky preconditioner (see Remark 5.9.8). The discretisation of (10.3) produces the system of linear equation $A\mathbf{u} = \mathbf{c}^{(i)}$. The number of degrees of freedom is 129^2. The stiffness matrix A and its $\mathcal{H}$-Cholesky factorisation (the preconditioner) are computed only once. Then for each $\mathbf{c}^i$, $i = 1, ..., i_{max}$, we perform the PCG method. The total computational times for $i_{max} = 10, 100, 1000$ are shown in Table 10.28. We denote the computational time of the algorithm "Leaves to Root" by t_1, the computational time of the algorithm "Root to Leaves" by t_2, the computational time of PCG by t_{Ch}. We see that for $i_{max} = 100$, $i_{max} = 1000$ HDD is much more efficient than

i_{max}	$t_1 + t_2$, sec.	t_{Ch}, sec.
10	38+2.8	29
100	38+27	117
1000	38+240	1048

Table 10.28: The total computational times of HDD and PCG for i_{max} right-hand sides.

PCG. Note that we compare the two-grid modification of HDD (steps $H = \frac{1}{128}$ and $h = \frac{1}{256}$) with the PCG method with one scale (step $h = \frac{1}{128}$). For the computational complexity and storage requirement of this HDD version, see Section 7.4.2. The HDD method with many right-hand sides can be applied, for example, to problems in electroencephalography and magnetoencephalography (EEG/MEG) or in Monte Carlo simulations.

10.11 Computing the Functionals of the Solution

Very often dimension of the initial multiscale problem is huge, the geometry is complex and not the whole solution is of interest, but the solution in a small subdomain $\omega \in \Omega$. Moreover it is interesting how the solution in ω changes when the right-hand side f and the Dirichlet boundary data g are changed. For this purposes one may compute different functionals of the solution (see Section 4.4.5). These functionals depend on f, g and the mappings Φ^g and Φ^f. Below we give an example.

10.11.1 Computing the Mean Value in $\omega \in T_{\mathcal{T}_h}$

In this section we realize the algorithm from Section 4.4.6 and compare the computed mean value of the solution u in a subdomain $\omega \in T_{\mathcal{T}_h}$ with the exact mean value in ω. We choose scales $H = 1/4$ and $h = 1/256$. The scale $H = 1/16$ gives a decomposition of the initial domain Ω into 16 cells (see Fig. 10.9). The problem to

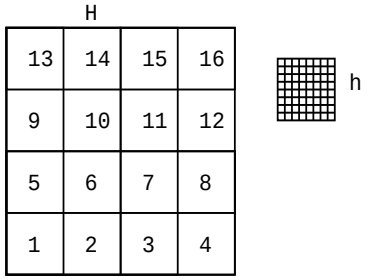


Figure 10.9: Model domain $\Omega = (0,1)^2$ and its subdivision into 16 subdomains.

be considered is:

$$\begin{array}{cc} -\operatorname{div}(\alpha(\mathbf{x})\nabla u) = 1 & \text{in } \Omega = (0,1)^2, \\ u = 0 & \text{on } \partial\Omega. \end{array}$$

We compute the exact mean value by the following formula:

$$\mu = \int_\omega u d\mathbf{x} = \sum_{t \in \mathcal{T}_h(\omega)} \frac{|t|}{3}(u_1 + u_2 + u_3),$$

where u_i, $i = 1, 2, 3$, is the solution computed by the PCG-method in vertices of the triangle t.
We compute the exact mean value μ and the approximate mean value μ_{HDD} (see Section 4.4.6) in each cell. Tables 10.29 and 10.30 compare these both values. We see from the second column (Tables 10.29, 10.30) that there are three patterns of subdomains: internal (6,7,10,11), angular (1,4,13,16) and border (2,3,5,8,9,12,14,15).

$\lvert\mu_{HDD} - \mu\rvert$	$\lvert\mu_{HDD} - \mu\rvert / \lvert\mu_{HDD}\rvert$	N of subdomain in Fig. 10.9
$1.73 * 10^{-3}$	$1.38 * 10^{-1}$	6
$1.71 * 10^{-3}$	$1.4 * 10^{-1}$	7
$1.71 * 10^{-3}$	$1.4 * 10^{-1}$	10
$1.71 * 10^{-3}$	$1.4 * 10^{-1}$	11
$1.71 * 10^{-3}$	$3.3 * 10^{-2}$	1
$1.71 * 10^{-3}$	$3.3 * 10^{-2}$	4
$1.71 * 10^{-3}$	$3.3 * 10^{-2}$	13
$1.71 * 10^{-3}$	$3.3 * 10^{-2}$	16
$1.71 * 10^{-3}$	$7 * 10^{-2}$	2
$1.71 * 10^{-3}$	$7 * 10^{-2}$	3
$1.70 * 10^{-3}$	$7 * 10^{-2}$	5
$1.71 * 10^{-3}$	$7 * 10^{-2}$	8
$1.71 * 10^{-3}$	$6.9 * 10^{-2}$	9
$1.71 * 10^{-3}$	$7 * 10^{-2}$	12
$1.71 * 10^{-3}$	$7 * 10^{-2}$	14
$1.71 * 10^{-3}$	$7 * 10^{-2}$	15

Table 10.29: Comparison of the approximate mean value with the exact mean value. $\alpha = 1.0/(1.001 + sin(50x)sin(50y))$, 257 dofs, $\|\tilde{\mathbf{u}}_\varepsilon - \tilde{\mathbf{u}}_{cg}\|_2 = 1.43 * 10^{-9}$, $\|\tilde{\mathbf{u}}_\varepsilon - \tilde{\mathbf{u}}_{cg}\|_\infty = 3.72 * 10^{-11}$.

$\lvert\mu_{HDD} - \mu\rvert$	$\lvert\mu_{HDD} - \mu\rvert / \lvert\mu_{HDD}\rvert$	subdomain in Fig. 10.9
$2.21 * 10^{-3}$	$1.38 * 10^{-1}$	6
$2.23 * 10^{-3}$	$1.38 * 10^{-1}$	7
$2.23 * 10^{-3}$	$1.4 * 10^{-1}$	11
$2.23 * 10^{-3}$	$1.39 * 10^{-1}$	10
$2.22 * 10^{-3}$	$3.4 * 10^{-2}$	1
$2.22 * 10^{-3}$	$3.4 * 10^{-2}$	4
$2.23 * 10^{-3}$	$3.4 * 10^{-2}$	13
$2.23 * 10^{-3}$	$3.4 * 10^{-2}$	16
$2.22 * 10^{-3}$	$7.06 * 10^{-2}$	2
$2.22 * 10^{-3}$	$7.07 * 10^{-2}$	3
$2.22 * 10^{-3}$	$7.04 * 10^{-2}$	8
$2.23 * 10^{-3}$	$7.07 * 10^{-2}$	5
$2.23 * 10^{-3}$	$7.07 * 10^{-2}$	9
$2.23 * 10^{-3}$	$7.07 * 10^{-2}$	12
$2.23 * 10^{-3}$	$7.07 * 10^{-2}$	14
$2.23 * 10^{-3}$	$7.07 * 10^{-2}$	15

Table 10.30: Comparison of the approximate mean value with the exact mean value. $\alpha = 1.0 + \frac{1}{2}sin(50x)sin(50y)$, 257 dofs, $\|\tilde{\mathbf{u}}_\varepsilon - \tilde{\mathbf{u}}_{cg}\|_2 = 7.97 * 10^{-10}$, $\|\tilde{\mathbf{u}}_\varepsilon - \tilde{\mathbf{u}}_{cg}\|_\infty = 2.37 * 10^{-11}$.

10.12 Conclusion to Numerics

By default we use the $\mathcal{H}$-Cholesky preconditioner in the PCG method.

1. The computational time of HDD is smaller than the time required for the direct $\mathcal{H}$-matrix inverse and slightly larger than the time of the PCG method (Tables 10.13, 10.14, 10.18, 10.24). But HDD solves problems with multiple right-hand side and multiple Dirichlet data faster than PCG does (Table 10.28).

2. For a smooth right-hand side f the HDD method with $f \in V_H \subset V_h$ (Sections 4.3.6, 7.4.2) is very efficient (Tables 10.28, 10.20). In fact, this modification of HDD requires less memory (Table 10.16) and smaller computational time than the PCG method for large numbers of degrees of freedom (Section 10.6).

3. The accuracy of the $\mathcal{H}$-matrix approximation increases with increasing the $\mathcal{H}$-matrix rank k (Tables 10.5, 10.6, 10.8, 10.9, 10.10, 10.11). Here it is important to have a balance between the discretisation error, the quadrature error and the $\mathcal{H}$-matrix approximation error because the total error depends on all of them.

4. The HDD method is stable for problems with oscillatory and jumping coefficients (Section 10.4). HDD is in the state to achieve the same accuracy as the accuracy of the exact PCG scheme (Tables 10.18, 10.25, 10.26). But the computation of the $\mathcal{H}$-Cholesky preconditioner is expensive or even impossible for a large number of degrees of freedom (Table 10.18).

5. The accuracy of the solution of problems with highly jumping coefficients can be improved (Tables 10.23, 10.24, 10.27) by decreasing the $\mathcal{H}$-matrix approximation error (by decreasing the parameter ε_a or by increasing the rank k).

6. HDD solves the simplified skin problem with highly jumping coefficients up to the discretisation error (Tables 10.25,10.26,10.27).

7. The HDD method may compute different functionals of the solution with small computational resources and a required accuracy (Tables 10.29, 10.30). If only the functionals are of interest, then there is no need to store the mappings Φ^g_ω and Φ^f_ω, $\omega \in T_{\mathcal{T}_h}$, and the storage requirements of HDD will even be fewer.

8. The computational time of HDD depends on many factors: a) the accuracy of the $\mathcal{H}$-matrix approximation, b) the ratio $\frac{H}{h}$, c) whether one or two grids are used, d) whether small scales are truncated or not.

11 Appendix

Lemma 11.0.1

$$\sum_{i=0}^{p} i^2 = \frac{1}{3}p^3 + \mathcal{O}(p^2).$$

Proof: Let $S(p) = \sum_{i=0}^{p} i^2 = ap^3 + bp^2 + cp + d$. Then

$$S(p+1) = \sum_{i=0}^{p+1} i^2 = S(p) + (p+1)^2.$$

We compare the corresponding coefficients in

$$ap^3 + bp^2 + cp + d + (p+1)^2 = a(p+1)^3 + b(p+1)^2 + c(p+1) + d.$$

and obtain $a = \frac{1}{3}$.

Lemma 11.0.2

$$\sum_{i=0}^{p} i2^i = (p-1)2^{p+1} + 2.$$

Proof: Consider

$$S(x) = \sum_{i=0}^{p} 2^{ix} = \frac{2^{x(p+1)} - 1}{2^x - 1}.$$

Then the derivative is

$$S'(x) = \sum_{i=0}^{p} i2^{ix} \log_e 2,$$

or

$$S'(x) = \left(\frac{2^{x(p+1)-1}}{2^x - 1}\right)' = \frac{(p+1)2^{x(p+1)} \log_e 2 - 2^x(2^{x(p+1)} - 1) \log_e 2}{(2^x - 1)^2}.$$

Comparing the expressions for the derivatives in $x = 1$

$$S'(1) = (p+1)2^{(p+1)} \log_e 2 - 2(2^{(p+1)} - 1) \log_e 2$$

and

$$S'(1) = \sum_{i=0}^{p} i2^i \log_e 2,$$

we obtain

$$(p+1)2^{(p+1)} \log_e 2 - 2(2^{(p+1)} - 1) \log_e 2 = \sum_{i=0}^{p} i2^i \log_e 2,$$

$$\sum_{i=0}^{p} i2^i = (p-1)2^{(p+1)} + 2.$$

■

Lemma 11.0.3

$$\sum_{i=0}^{p}(p-i)2^i = 2^{p+1} - p - 2.$$

Proof:

$$\sum_{i=0}^{p}(p-i)2^i = p\sum_{i=0}^{p}2^i - \sum_{i=0}^{p}i2^i = p2^{p+1} - p - (p-1)2^{p+1} - 2 = 2^{p+1} - p - 2.$$

Lemma 11.0.4

$$\sum_{i=0}^{p}(p-i)^2 2^i = 3 \cdot 2^{p+1} + \mathcal{O}(p^2).$$

Proof:

$$\sum_{i=0}^{p}(p-i)^2 2^i = \sum_{i=0}^{p}(p^2 2^i - 2pi2^i + i^2 2^i)$$

$$\leq p^2 2^{p+1} - 2p(p-1)2^{p+1} + (p^2 - 2p + 3)2^{p+1} = 3 \cdot 2^{p+1} + \mathcal{O}(p^2).$$

Lemma 11.0.5

$$\sum_{i=0}^{p} i^2 2^i = (p^2 - 2p + 3)2^{p+1} - 6.$$

Conclusion

The hierarchical domain decomposition method is a flexible tool for solving 2D elliptic boundary value problems with L^∞ coefficients. HDD computes the hierarchical solution operators $\mathcal{F}_h$ and $\mathcal{G}_h$ and allows the representation of the FE solution of the initial problem in the form

$$u_h = \mathcal{F}_h f_h + \mathcal{G}_h g_h, \tag{11.1}$$

where f_h is the FE right-hand side, and g_h is the FE Dirichlet boundary data. The operators $\mathcal{F}_h$ and $\mathcal{G}_h$ are efficiently approximated in the $\mathcal{H}$-matrix format.
Representation (11.1) allows HDD to solve problems with multiple right-hand side and multiple Dirichlet data with reduced computational costs.
HDD may compute different functionals of the solution (the solution on the skeleton or at a single point and a mean value $\int_\omega u_h dx$, $\omega \subset \Omega \subset \mathbb{R}^2$, or a flux $\oint \nabla u \overrightarrow{n} dx$ etc.) with less resources.

The application of the $\mathcal{H}$-matrix technique to HDD (see Chapter 5) results in the computational cost $\mathcal{O}(k^2 n_h \log^3 n_h)$ and the storage cost $\mathcal{O}(k n_h \log^2 n_h)$, where n_h is the number of degrees of freedom on a fine scale. In the case of two grids the estimates are $\mathcal{O}(k^2 \sqrt{n_h n_H} \log^3 \sqrt{n_h n_H})$ and $\mathcal{O}(k \sqrt{n_h n_H} \log^2 \sqrt{n_h n_H})$, respectively, where n_H is the number of degrees of freedom on a coarse scale.
The accuracy of the $\mathcal{H}$-matrix approximation depends on the maximal rank k.
The cost of solving homogeneous problems ($f_h \equiv 0$) is $\mathcal{O}(k^2 n_h)$, i.e., linear because in this case the HDD method does not compute the more expensive discrete operator $\mathcal{F}_h$ in (11.1). Thus, only the operator $\mathcal{G}_h$ is computed.

HDD was successfully applied to problems with the right-hand side from a coarser subspace $V_H \subset V_h$, to problems with strongly oscillatory coefficients and to problems with highly jumping coefficients.

As we mentioned in Chapter 8, the hierarchical background of the HDD method provides its effective parallelization. For a machine with $q = 2^r$ processors, the parallel complexity of the algorithm "Leaves to Root" is estimated (Lemma 8.3.3) by

$$\frac{C' k^2 \sqrt{n_h} \log^2 \sqrt{n_h} + \tilde{C} k^2 n_h}{q^{0.45}} + C''(1 - \frac{3^r}{4^r}) \sqrt{n_h} n_{min}^2 + C k^2 \frac{n_h}{q} \log^3 \frac{n_h}{q}$$

where $C', \tilde{C}, C'', C \in \mathbb{R}_+$.
The parallel complexity of the algorithm "Root to Leaves" on a machine with q processors is estimated (Lemma 8.3.6) by

$$C k^2 \frac{n_h}{q} \log^2 \frac{n_h}{q} + \frac{28 k \sqrt{n_h}}{q^{0.45}}, \quad C \in \mathbb{R}_+.$$

Future work

1. *Further optimization.* In regards to the $\mathcal{H}$-matrix conversion, the computational complexity $\mathcal{O}(n \log^2 n)$ can be reduced to $\mathcal{O}(n \log n)$. This, in turn, can be shown to reduce the computational complexity of HDD to $\mathcal{O}(n \log^2 n)$.

2. *3D case.* The application of the HDD method to 3D problems is possible. The differences with the 2D case are:
   - For the approximation of the mapping Ψ^g_ω, $\omega \in T_{\mathcal{T}_h}$, the standard admissibility condition should be applied instead of the weak one. This transforms the approximation of the mapping Φ^g_ω to an $\mathcal{H}$-matrix, instead of a low-rank matrix as in 2D.
   - Since constants in the complexity estimates of the $\mathcal{H}$-matrix technique are dependent on the spatial dimension, the constants in estimates of the computational complexities and storage requirements of HDD in 3D will be larger compared to those in 2D, yet almost linear.
   - The data structures in the implementation must be modified.

3. *Parallelization.* For a parallel implementation of HDD, the recent work [44] can be used.

Bibliography

[1] Basic linear algebra subroutines. *www.netlib.org/blas/*.

[2] Linear algebra package. *www.netlib.org/lapack/*.

[3] Assyr Abdulle and Weinan E. Finite difference heterogeneous multi-scale method for homogenization problems. *J. Comput. Phys.*, 191(1):18–39, 2003.

[4] Assyr Abdulle and Christoph Schwab. Heterogeneous multiscale FEM for diffusion problems on rough surfaces. *Multiscale Model. Simul.*, 3(1):195–220 (electronic), 2004/05.

[5] T. Alarcón, H. M. Byrne, and P. K. Maini. A multiple scale model for tumor growth. *Multiscale Model. Simul.*, 3(2):440–475 (electronic), 2005.

[6] Grégoire Allaire and Robert Brizzi. A multiscale finite element method for numerical homogenization. *Multiscale Model. Simul.*, 4(3):790–812 (electronic), 2005.

[7] Ivo Babu**v**ska. Homogenization and its application. Mathematical and computational problems. In *Numerical solution of partial differential equations, III (Proc. Third Sympos. (SYNSPADE), Univ. Maryland, College Park, Md., 1975)*, pages 89–116. Academic Press, New York, 1976.

[8] P. Bastian. *Numerical computation of multiphase flow in porous media*. Habilitationsschrift, Heidelberg, Germany, 1999.

[9] M. Bebendorf. Hierarchical LU decomposition-based preconditioners for BEM. *Computing*, 74(3):225–247, 2005.

[10] M. Bebendorf. Why approximate lu decompositions of finite element discretisations of elliptic operators can be computed with almost linear complexity. *Technical report in Max-Planck-Institut MIS www.mis.mpg.de, Leipzig, Germany*, 8, 2005.

[11] M. Bebendorf and W. Hackbusch. Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. *Numer. Math.*, 95(1):1–28, 2003.

[12] Lions J-L. Bensoussan, A. and G. Papanicolaou. Asymptotic analysis for periodic structures. 1978.

[13] S. Börm, L. Grasedyck, and W. Hackbusch. *Hierarchical Matrices*, volume 21 of *Lecture Note*. Max-Planck Institute for Mathematics, Leipzig, 2003. www.mis.mpg.de.

[14] Dietrich Braess. *Finite elements*. Cambridge University Press, Cambridge, second edition, 2001. Theory, fast solvers, and applications in solid mechanics, Translated from the 1992 German edition by Larry L. Schumaker.

[15] Zhiming Chen and Thomas Y. Hou. A mixed multiscale finite element method for elliptic problems with oscillating coefficients. *Math. Comp.*, 72(242):541–576 (electronic), 2003.

[16] Philippe G. Ciarlet. *The finite element method for elliptic problems*. North-Holland Publishing Co., Amsterdam, 1978. Studies in Mathematics and its Applications, Vol. 4.

[17] Doina Cioranescu and Jeannine Saint Jean Paulin. *Homogenization of reticulated structures*, volume 136 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 1999.

[18] Weinan E and Bjorn Engquist. The heterogeneous multiscale methods. *Commun. Math. Sci.*, 1(1):87–132, 2003.

[19] Weinan E and Bjorn Engquist. Multiscale modeling and computation. *Notices Amer. Math. Soc.*, 50(9):1062–1070, 2003.

[20] Weinan E, Xiantao Li, and Eric Vanden-Eijnden. Some recent progress in multiscale modeling. In *Multiscale modelling and simulation*, volume 39 of *Lect. Notes Comput. Sci. Eng.*, pages 3–21. Springer, Berlin, 2004.

[21] Jens Eberhard. Upscaling und mehrgitterverfahren für strömungen in heterogenen porösen medien. *Ph.D. Thesis, University of Heidelberg, Germany*, 2003.

[22] Björn Engquist, Per Lötstedt, and Olof Runborg, editors. *Multiscale methods in science and engineering*, volume 44 of *Lecture Notes in Computational Science and Engineering*. Springer-Verlag, Berlin, 2005. Papers from the conference held in Uppsala, January 26–28, 2004.

[23] Bjorn Engquist and Erding Luo. Convergence of a multigrid method for elliptic equations with highly oscillatory coefficients. *SIAM J. Numer. Anal.*, 34(6):2254–2273, 1997.

[24] Alan George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10:345–363, 1973. Collection of articles dedicated to the memory of George E. Forsythe.

[25] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[26] I.G. Graham, P. Lechner, and R. Scheichl. Domain decomposition for multiscale pdes. *Bath Institute for Complex Systems, Preprint 11, www.bath.ac.uk/math-sci/BICS*, 2006.

[27] L. Grasedyck. Theorie und anwendungen hierarchischer matrizen. *Ph.D. Thesis, University of Kiel, Germany*, 2001.

[28] L. Grasedyck and S. Börm. $\mathcal{H}$-matrix library: www.hlib.org.

[29] L. Grasedyck and W. Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70(4):295–334, 2003.

[30] Khoromskij B. N. Hackbusch, W. A sparse $\mathcal{H}$-matrix arithmetic. II. Application to multi-dimensional problems. *Computing*, 64(1):21–47, 2000.

[31] W. Hackbusch. *Elliptic differential equations*, volume 18 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1992. Theory and numerical treatment, Translated from the author's revision of the 1986 German original by Regine Fadiman and Patrick D. F. Ion.

[32] W. Hackbusch. *Iterative solution of large sparse systems of equations*, volume 95 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 1994. Translated and revised from the 1991 German original.

[33] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[34] W. Hackbusch. $\mathcal{H}$-matrix techniques and multi-scale problems. *Workshop: Numerical methods for multiscale Problems, MPIMS, Leipzig, Germany*, 2002.

[35] W. Hackbusch. Direct domain decomposition using the hierarchical matrix technique. In *Domain decomposition methods in science and engineering*, pages 39–50 (electronic). Natl. Auton. Univ. Mex., México, 2003.

[36] W. Hackbusch. *Hierarchical Matrizen - Algoritmen und Analysis*, volume 22. Max-Planck-Institut fr Mathematik, Leipzig, 2004. www.mis.mpg.de.

[37] W. Hackbusch, B. N. Khoromskij, and R. Kriemann. Hierarchical matrices based on a weak admissibility criterion. *Computing*, 73(3):207–243, 2004.

[38] W. Hackbusch, B.N. Khoromskij, and R. Kriemann. Direct Schur complement method by domain decomposition based on $\mathcal{H}$-matrix approximation. *Comput. Vis. Sci.*, 8(3-4):179–188, 2005.

[39] Viet Ha Hoang and Christoph Schwab. High-dimensional finite elements for elliptic problems with multiple scales. *Multiscale Model. Simul.*, 3(1):168–194 (electronic), 2004/05.

[40] Thomas Y. Hou and Xiao-Hui Wu. A multiscale finite element method for elliptic problems in composite materials and porous media. *J. Comput. Phys.*, 134(1):169–189, 1997.

[41] Thomas Y. Hou, Xiao-Hui Wu, and Zhiqiang Cai. Convergence of a multiscale finite element method for elliptic problems with rapidly oscillating coefficients. *Math. Comp.*, 68(227):913–943, 1999.

[42] V. V. Jikov, S. M. Kozlov, and O. A. Ole**ı**nik. *Homogenization of differential operators and integral functionals*. Springer-Verlag, Berlin, 1994. Translated from the Russian by G. A. Yosifian.

[43] B.N Khoromskij and A. Litvinenko. Domain decomposition based $\mathcal{H}$-matrix preconditioner for the skin problem in 2d and 3d. *Max-Planck Institute, online preprint www.mis.mpg.de/preprints*, (95), 2006.

[44] R. Kriemann. Parallele algorithmen für $\mathcal{H}$-matrizen. *Ph.D. Thesis, University of Kiel, Germany*, 2004.

[45] R. Kriemann. Parallel $\mathcal{H}$-matrix arithmetics on shared memory systems. *Computing*, 74(3):273–297, 2005.

[46] S. Le Borne and L. Grasedyck. $\mathcal{H}$-matrix preconditioners in convection-dominated problems. *SIAM J. Matrix Anal. Appl.*, 27(4):1172–1183 (electronic), 2006.

[47] S. Le Borne, L. Grasedyck, and R. Kriemann. Parallel black box domain decomposition based $\mathcal{H} - lu$ preconditioning. *Max-Planck-Institut MIS, Leipzig, www.mis.mpg.de*, Preprint 115:(electronic), 2005.

[48] M. Lintner. The eigenvalue problem for the 2D Laplacian in $\mathcal{H}$-matrix arithmetic and application to the heat and wave equation. *Computing*, 72(3-4):293–323, 2004.

[49] A. Litvinenko. Documentation for the HDD method. *Technical report in Max-Planck-Institut MIS, www.mis.mpg.de/preprints/tr/index.html, Leipzig, Germany*, 5, 2006.

[50] Pingbing Ming and Xingye Yue. Numerical methods for multiscale elliptic problems. *J. Comput. Phys.*, 214(1):421–445, 2006.

[51] L. Mirsky. Symmetric gauge functions and unitarily invariant norms. *Quart. J. Math. Oxford Ser. (2)*, 11:50–59, 1960.

[52] N. Neuss, W. Jäger, and G. Wittum. Homogenisierung und mehrgitter. *Ph.D. Thesis, Universiteat of Heidelberg, Heidelberg, Germany*, 1995.

[53] N. Neuss, W. Jäger, and G. Wittum. Homogenization and multigrid. *Computing*, 66(1):1–26, 2001.

[54] Jeff Ovall. Duality-based adaptive refinement for elliptic pdes. *Ph.D. Thesis, University of California at San Diego*, 2004.

[55] Peter J. Park and Thomas Y. Hou. Multiscale numerical methods for singularly perturbed convection-diffusion equations. *International Journal of Computational Methods*, 1(1):17–65, 2004.

[56] A.V. Skvorcov. Review of algorithms for delaunay triangulation building. *Computation methods and programming*, T.3, 2002.

Index

Lebenslauf

Alexander Litvinenko

Geboren am 31.08.1979 in Almaty, Kasachstan
1986 bis 1994 Besuch der Schule 90 in Almaty.
1994 bis 1995 Besuch der Physikalische-Mathematische Schule von NSU in Akademgorodok, Novosibirsk, Abitur 1996.
1996 bis 2000 Studium an der Staatlichen Universität Novosibirsk (NSU).
2000 Bachelor Diplom in Mathematik.
2000 bis 2002 Studium an der Staatlichen Universität Novosibirsk und Sobolev Institut für Mathematik.
2002 Magister Diplom in Mathematik.
2002 bis 2006 Doktorand am Max-Planck-Institut für Mathematik in den Naturwissenschaften.

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialen oder erbrachten Dienstleistungen als solche gekennzeichnet.

10.04.2006

(Unterschrift)