Technische Universität München
Lehrstuhl für Computation in Engineering

Towards Massive Parallel Fluid Flow Simulations in Computational Engineering

Jérôme Frisch

Vollständiger Abdruck der von der Ingenieurfakultät Bau Geo Umwelt der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. habil. Michael Manhart

Prüfer der Dissertation:

1. Univ.-Prof. Dr. rer. nat. Ernst Rank
2. Univ.-Prof. Dr.-Ing. habil. Christoph van Treeck
   Rheinisch-Westfälische Technische Hochschule Aachen
3. Prof. Dr. Sci. Takayuki Aoki
   Tokyo Institute of Technology / Japan

Die Dissertation wurde am 03.07.2014 bei der Technischen Universität München eingereicht und durch die Ingenieurfakultät Bau Geo Umwelt am 30.09.2014 angenommen.

Zusammenfassung

Da Rechner über die letzten Jahre immer schneller und leistungsfähiger geworden sind, können heutzutage Probleme aus dem Ingenieurbereich gelöst werden, die vor zehn Jahren noch undenkbar gewesen wären. Solch ein Lösungskonzept setzt allerdings den Einsatz von massiv parallelen Systemen voraus, die wiederum einige Herausforderungen in Punkto Programmierung und Effizienz darstellen.

Diese Arbeit stellt ein Gesamtkonzept vor, wie ein Simulationsablauf effizient auf einem System wie SuperMUC, einem von Deutschlands drei Höchstleitungsrechnern, effizient umgesetzt werden kann. Nach einer mathematischen Einführung der zugrundeliegenden Navier-Stokes Gleichungen für ein inkompressibles Newtonsches Fluid werden numerische Methoden und im Speziellen die räumliche und zeitliche Diskretisierung erläutert.

Das Rückgrat der gesamten Simulation bildet dabei die eingesetzte Datenstruktur, die auf einem adaptiven, hierarchischen, block-strukturierten, kartesischen, orthogonalen Gitteransatz beruht. Die verwendeten Kommunikationsroutinen sowie der Datenaustausch in der hierarchischen Struktur werden im Detail dargelegt und ein Ansatz zum Lastausgleich auf verteilten Prozessen, der unverzichtbar für eine parallel verteile Berechnung ist, wird im Detail erläutert.

Der beschriebene numerische Ansatz beruht auf einer Druckkorrekturmethode basierend auf der Chorinprojektion zur Trennung der Berechnung der Geschwindigkeiten und des Drucks, sodass zu jedem Zeitschritt eine Druck-Poissongleichung gelöst werden muss. Diese wird mit Hilfe eines tief in der vorgestellten Datenstruktur integrierten mehrgitterähnlichen Ansatzes gelöst, für welchen Genauigkeitsstudien sowie Leistungsmessungen für bis zu 80 Milliarden Druckwerte auf SuperMUC durchgeführt worden sind.

Weiterhin wird eine interaktive Visualisierungmöglichkeit vorgestellt, die auf der vorbezeichneten Datenstruktur beruht und es ermöglicht, enorm große Datenmengen zu verarbeiten und darzustellen. Hierzu kann entweder ein kleiner Teilbereich in großer Auflösung betrachtet oder ein Gesamtüberblick mit vergröberten Daten gewonnen werden.

Berechnungen zum Vergleich von Simulationsergebnissen mit Literaturwerten für Standard-Benchmarks wie die Driven Cavity, den Strömungskanal, die von Kármán Wirbelstraße, die nischengetriebene thermische Strömung oder die Rayleigh-Bénard Strömung liefern ausgezeichnete Übereinstimmungen. Weiterhin wurde ein komplexes Beispiel aus dem Ingenieur-

bereich durchgerechnet. Hierzu wurde eine zonale Ganzjahressimulation für einen Raum und einen Menschen durchgeführt, der mit einem Thermoregulationsmodell gekoppelt ist. Zu einem bestimmten vordefinierten Zeitpunkt wird ein detaillierter Schnappschuss des Raumes mittels der vorgestellten CFD Simulation erstellt, um das genaue Geschwindigkeitsprofil im Raum zu ermitteln und weitere Möglichkeiten zur Auswertung des thermischen Komforts zu erhalten.

In dieser Arbeit werden Möglichkeiten aufgezeigt, Methoden und Lösungsansätze der Informatik und insbesondere des Hoch- und Höchstleitungsrechnens effektiv und effizient auf ingenieurwissenschaftliche Fragestellungen anzuwenden, um damit komplexe Probleme handhabbar zu machen, die bisher nur durch Methoden wie Dimensionsreduktion vereinfacht betrachtet werden konnten. Die dabei entstandenen Synergieeffekte zwischen beiden Disziplinen sind evident und äußerst vielversprechend, da nun auf der Ingenieurseite ein wesentlich tieferer Einblick in die Problemstellung möglich ist – etwa im Fall der gekoppelten thermischen Strömungssimulation mit einem Menschmodell – sowie auf der Informatikseite nunmehr methodologische Konzepte wie der vorgestellte mehrgitterähnliche Ansatz weiter ausgebaut und optimiert werden konnten, um in ingenieurbasierten Anwendungen effizient genutzt werden zu können.

Abstract

The trend for computers to get faster and faster means it is now possible to simulate many larger problems from an engineering-based domain, such as complex indoor air flow scenarios driven by buoyancy, which were deemed unsolvable a decade ago. Using a massive parallel approach on a supercomputer poses some challenges which have to be tackled before initial results can be achieved, however.

This work presents a simulation pipeline which is able to work efficiently on massive parallel systems such as SuperMUC, one of Germany's national supercomputers. After describing the mathematical formulation of an incompressible Newtonian fluid flow based on the Navier-Stokes equations, the standard numerical methods such as the finite difference and finite volume based approaches are discussed.

The newly introduced data structure forms the backbone of the complete system and consists of an adaptive, hierarchic, block-structured, orthogonal, Cartesian grid structure. The communication routines for data exchange in the hierarchical structure are highlighted. Furthermore, a load balancing concept, which is essential for massive parallel approaches, is analysed and described in detail.

A pressure correction method based on Chorin's projection method is applied to decouple the velocity and pressure fields, resulting in a Poisson equation for the pressure which has to be solved in every time step. An efficient method based on a multi-grid-like solver and integrated into the data structure is presented, and accuracy as well as performance measurements of the data structure are presented for up to 80 billion pressure unknowns solved on SuperMUC.

Furthermore, an interactive visualisation technique is introduced, which is capable of selecting only a small detail of the entire domain in fine resolution or the complete overview in a coarser resolution, in order to handle the massive amount of data generated by the type of computation mentioned above.

Standard benchmarks are presented, which show the excellent agreement of the simulation results and literature values for the most common examples, such as a lid-driven cavity, a channel flow, a von Kármán vortex street, heated side walls, and a Rayleigh-Bénard convection.

Last but not least, a complex example of a human manikin in a test room is simulated by a zonal computation coupled with the CFD simulation presented in order to obtain a detailed

snapshot of the velocity distribution in the room together with a detailed thermoregulation model for evaluating the temperatures on the surface of the manikin.

This work highlights possibilities to efficiently combine methods from computer science – especially from the field of high-performance computing – with problems from engineering-based domains in order to leverage and facilitate the treatment of complex problems which have not been able to be solved directly until now. The synergy effects between the two disciplines are obvious and promising. On the engineering side, a deeper insight into the problem itself can be gained, for a coupled thermal simulation with a human manikin model, for example. Furthermore, on the computer science side, the methodological concepts such as the proposed multi-grid-like solver concept can be expanded and optimised in order to be used efficiently in engineering-based applications.

Preface

This thesis was created during my employment at the Chair for Computation in Engineering at Technische Universität München. I would like to express my deep gratitude to some people who supported my work which finally led to the thesis at hand.

First of all, I would like to thank my doctoral supervisor Prof. Dr. rer. nat. Ernst Rank for his support of my work and for all the arising opportunities during the course of my thesis. Under his tutelage, I could not only improve my scientific knowledge and approaches, but also gather expertise in other areas such as teaching, administrative competence, or organisational skills. By granting me great room for my own development and progress, but also guiding my research during fruitful discussions, he understood to create an outstanding working climate at his chair providing more than enjoyable working conditions.

Many thanks go also to Prof. Dr.-Ing. habil. Christoph van Treeck for sparking my interest in building physics, computational fluid dynamics, and thermal comfort assessment as well as for being advisor to my thesis. His ideas and comments regarding my work during the time he was at Fraunhofer Institute for Building Physics (IBP) in Holzkirchen and during the time he was heading the Chair for Energy Efficient and Sustainable Building E3D at RWTH Aachen have proven invaluable.

Additionally, I would like to thank Prof. Takayuki Aoki Dr. Sci. very much for harbouring me during my international stay abroad at the Global Scientific Information and Computing Center at Tokyo Institute of Technology in Japan. The experience abroad and the intercultural exchange as well as the scientific exchange with his staff and himself sparked a whole range of new ideas which I proudly incorporated into the work at hand. I also want to express my gratitude for his acceptance of being an international advisor to my thesis.

I would also like to thank Prof. Dr.-Ing. habil. Michael Manhart for acting as chairman during the formal procedures of my dissertation.

A very great thanks goes to PD Dr. rer. nat. Ralf-Peter Mundani who endured a lot during his time as head of the 'Efficient Algorithms' group which I was a proud member of. Amongst others, countless discussions during scientific meetings at the 'Bügelbrett' helped shaping this work in more ways than one might think. Moreover, his striving to use more and more computing cores pushed me ultimately to prepare the code for a full SuperMUC run.

Furthermore, I would like to thank Dr.-Ing. Andreas Niggl and Prof. Dr.-Ing. habil. Alexander Düster for recruiting me as student assistant for working at the Chair for Computation in Engineering in their projects. Both are mainly responsible for helping to develop many useful skills during that very instructive time.

Likewise, I want to thank all my other colleagues and former colleagues at the Chair for Computation in Engineering not only for their numerous inputs and comments to my work,

but also for their nice company over the course of six years spent in the third floor of the main building at TUM. Amongst others, special thanks goes (in alphabetical order) to Dr.-Ing. Milica Grahovac, Dr. rer. nat. Jovana Knežević, Prof. Dr.-Ing. Petra Liedl, Nevena Perović M.Sc., Dr.-Ing. Michael Pfaffinger, Dr. rer. nat. Vasco Varduhn, and Nils Zander M.Sc. Furthermore, I would like to especially thank Hanne Cornils, our secretary who helped with all administrative questions and troubles, as well as with cookies in times of dire need.

Additionally, I am very grateful for the support and usage of the computing resources at KAUST, at UVT, and at LRZ whilst working on various research projects. Especially during the 'Extreme Scaling Workshop' held in June 2014 at LRZ, I had the opportunity to use the entire SuperMUC exclusively, which has proven invaluable for analysing the scaling behaviour of the code created for this thesis.

Last, but not least, I would like to express my sincerest gratitude to my family and friends, especially to my parents and to my partner Hanna for their continuous and unconditional support and love, and their huge compromises regarding 'time management'.

München, October 2014

Jérôme Frisch
frisch@tum.de

Contents

1. Introduction and Motivation

The massively increased performance in computing power and resources means that it is now possible to tackle huge problems in the area of science and engineering which were deemed unsolvable a decade or even a few years ago. The PITAC report [BL05] states on page 2 in its principal findings that '*computational science is now indispensable to the solution of complex problems in every sector, from traditional science and engineering domains to such key areas as national security, public health, and economic innovation. Advances in computing and connectivity make it possible to develop computational models and capture and analyze unprecedented amounts of experimental and observational data to address problems previously deemed intractable or beyond imagination.*'

Back in the mid 60s, Gordon E. Moore, co-founder of Intel Corporation, stated that '*the number of transistors incorporated in a chip will approximately double every 24 months*'[1] based on extrapolations of the number of integrated circuit components from 1959 to 1965 as shown in Figure 3 of his paper [Moo65]. His extrapolations proved right and Dr. Carver A. Mead, a Professor at Cal Tech, dubbed this observation henceforth *Moore's Law*. Even nowadays, his observation remains valid and can be seen clearly when depicting the development in performance of supercomputers as presented on a half-year basis at the International Supercomputing Conference (ISC) in June and the ACM/IEEE Supercomputing Conference (SC) in November (published on `http://top500.org` for instance).

With this observation of the ever-increasing performance of computing resources, experts estimate the *exa-scale era* – i. e. 10^{18} floating point operations per second (flops) and more – will be reached around 2018, and great efforts are being made worldwide to tackle the upcoming challenge. To this end, the German Research Foundation DFG initiated a priority programme SPP 1648 called 'Software for Exascale Computing' (code name: SPPEXA) with the dedicated mission statement to address '*fundamental research on the various aspects of High-Performance Computing (HPC) software, which is particularly urgent against the background that we are currently entering the era of ubiquitous massive parallelism.*'[2]

These efforts show the need for adequate solution techniques and software for handling problems from an engineering-based environment such as indoor airflow or thermal comfort simulations using a massive parallel computing approach specifically designed for supercomputers.

[1] Source: `http://www.intel.com/content/www/us/en/history/museum-gordon-moore-law.html`

[2] Source: `http://www.dfg.de/foerderung/info_wissenschaft/archiv/2011/info_wissenschaft_11_59/index.html`

Unfortunately, the PITAC report summarises the findings of Chapter 4 on page 36 by stating that '*today's computational science ecosystem is unbalanced, with a software base that is inadequate to keep pace with and support evolving hardware and application needs. By starving research in both enabling software and applications, the imbalance forces researchers to build atop inadequate and crumbling foundations rather than on a modern, high-quality software base. The result is greatly diminished productivity for both researchers and computing systems.*'

Figure 1.1.: SuperMUC - German national supercomputer installed at the Leibniz Supercomputing Centre (LRZ) of the Bavarian Academy of Sciences and Humanities (courtesy of LRZ; see also Appendix B.2.3).

Today's most widely used commercial CFD (Computational Fluid Dynamics) programs are not designed for massive parallel usage on supercomputers such as SuperMUC, one of Germany's national supercomputers installed in 2012 and depicted in Figure 1.1. Furthermore, even some of the well-established 'cutting-edge' research codes from academia are not able to run on today's existing supercomputers or even larger clusters, not to mention future generations of computing systems.

A further degree of complexity is added by the current hardware construction of supercomputers which is becoming more and more heterogeneous. According to the Top500 list, before November 2005 no computer used (modern) accelerators such as general purpose graphics processing units (GPGPU). In recent years, however, they have been installed more frequently and according to the list of November 2013, 53 out of 500 systems used accelerators (22 systems with NVIDIA Fermi, 15 systems with NVIDIA Kepler, 12 systems with Intel Xeon Phi, and 4 others). Thus, in November 2013, only 10% of the machines used accelerators, but they accounted for 35% of the total agglomerated performance of the Top500 list. This trend can be seen clearly by looking at *Stampede*[3], number 7 in the ranking of November 2013 with a peak performance of 9.6 peta flops, where 'only' 2.2 peta flops come from the general CPUs (Intel E5 8-core 'Sandy Bridge' processors) and 7.4 peta flops come

[3] `https://www.tacc.utexas.edu/stampede`

from accelerators (Intel Xeon Phi 61-core 'Knights Corner' co-processors). Furthermore, in the second expansion phase of SuperMUC, it is also planned to upgrade the system with Intel Xeon Phi co-processors to increase the performance[4].

Code specifically developed for homogeneous supercomputers has to be rewritten again in order to (efficiently) use the heterogeneous accelerator-based hardware. This task is not tackled in this work, but is mentioned merely to indicate the future trends, and will be kept in mind in the design of the parallel structure.

Focus of this Work

This work highlights the overall design process of a Navier-Stokes-based fluid flow simulation capable of running on massive parallel systems such as SuperMUC, i. e. non-accelerator-based, homogeneous, distributed computing environments. A cornerstone is the specifically dedicated data structure, capable of organising the distributed data in a hierarchic manner for simulation as well as visualisation purposes. Performance measurements as well as validations using standard benchmarks are shown.

The goal of this thesis is not to introduce a completely new numerical scheme for a highly accurate flow computation, however, but rather to use a very simple computational kernel-based on the incompressible Navier-Stokes equations with an explicit time stepping scheme. On the basis of this kernel, the data structure is introduced and explained in detail, and should pave the way for future implementations of higher-order methods using the aforementioned highly efficient data structure.

Last but not least, a complex example from engineering practice is presented by coupling the present code with other engineering-based approaches for computing whole-year simulations of buildings, for example.

Comparison to the State of the Art

As briefly mentioned above, most CFD research codes do not run effectively (or do not run at all) on massive parallel systems. There are, of course, CFD codes which run efficiently on (massive) parallel systems and were specially designed for this purpose, such as WaLBerla [FGD+09], Virtual Fluids [GKT+06], Multilevel Communicating Interface code [GK10], or SciRUN [PJ95], for example. All have their special field of interest and focus, and are designed and based on different principles. While WaLBerla and Virtual Fluids use Lattice-Boltzmann-based simulations for fluid flow computations, [GK10] uses a Navier-Stokes approach, and SciRUN is a general problem-solving environment with an interchangeable solver

[4]Source: `http://www.lrz.de/services/compute/supermuc/systemdescription`

back-end which is focussed on a computational steering-based approach for real-time feedback. Further information on computational steering-based approaches as well as interactive data exploration can be found in [Mun13], for example.

In terms of open source software, OpenFOAM should be mentioned, which '*has an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to solid dynamics and electromagnetics*'[5]. As it is 'open' it can be modified and extended by users and adapted to their specific needs. Unfortunately, the underlying data structure is widespread throughout all functionalities from pre-processing and the simulation kernels to post-processing and, thus, a change in the data structure in order to tune it for massive parallel computing scenarios is a massively invasive procedure and, hence, deemed infeasible for the work presented here.

It is shown that an adaptive data structure is indispensable for the simulation of large scenarios with different levels of interest, as a uniform distribution to the lowest level of interest results in an exorbitant number of unknowns and might not be solvable even with today's supercomputer resources.

Adaptive grids are already well-studied in literature. This work's data structure is based on adaptive, non-overlapping, orthogonal block-structured Cartesian grids, which is similar to the concept described by Manhart and Wengle [MW94] using a stepwise approximation with regular grids. A different approach studied by [Had05], for example, is the so-called Chimera approach, where different grids of different resolutions overlap and resolve the region of interest. Yet another possibility is the use of so-called hierarchical hybrid grids (HHG) as proposed by [GKSR14], which are composed from unstructured tetrahedral finite elements that are regularly refined. A further, well-organised study of applied data structures and grids can be found in [Meh10], for example.

The major difference between the data structure applied in this work and other regular data structures used in the codes above is that the parent grids are not removed when a refinement is applied, but kept in the system. Although this increases the memory footprint, the parent grids are essential for the solving mechanism of the CFD equations, which is deeply integrated into the data structure itself. Details about this crucial point are given in Section 3.8 and 4.6.

Furthermore, the coarsened, interpolated information on the (not deleted) parent levels can be used immediately for interactive visualisation purposes. Given a grid distribution with billions of unknowns, an 'on-the-fly' visualisation of the complete domain is no longer possible, because the network bandwidth as well as the visualisation components limit the maximal number of unknowns which current state of the art visualisation systems can display. Thus, the data structure provides information on a coarser level, which can be used for a full view, even if the simulation itself runs on a finer data set. While zooming in, the bandwidth and number of unknowns can be kept constant in order to have a smooth visualisation experience during the complete zooming process. This concept is explained in detail in Chapter 5.

[5] Source: `http://www.openfoam.org/features`

As the application to engineering-based simulations such as indoor air flow and thermal comfort evaluation is of interest here, it should be mentioned that the state of the art approach for solving large problems is nowadays a co-simulation based on a 1D to 3D coupling [SvTB+11]. Several heterogeneous computational codes can be coupled using a scale-adaptive approach. The code presented can be interfaced to such a coupling and deliver indoor air flow specific data such as velocities and temperatures in high resolution, while a customisation of the underlying data structure is easily possible as shown in Chapter 7.

Detailed Simulation Pipeline of a CFD Program and Outline of the Work

Computational Fluid Dynamics (CFD) simulations are used nowadays in various aspects of life. Starting from optimised car or aircraft designs and flood simulations of complete landscapes to urban air flow simulations, a large number of codes for various fields have been developed over recent decades. Regardless of their usage, all codes suffer from the same problems: they have to deal with a huge, resolved computational domain in order to capture all relevant physical effects, and they require a large amount of computational resources.

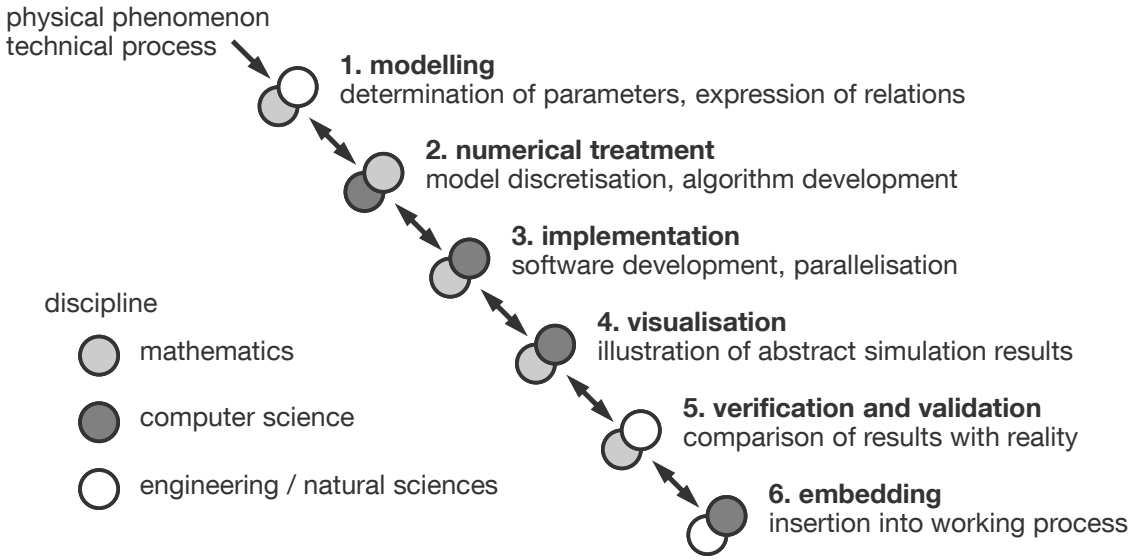


Figure 1.2.: Numerical simulation: from phenomena to predictions. (adapted from [BHW02])

In order to construct a program package capable of simulating fluid flow, at least five steps depicted in Figure 1.2 have to be performed. All these steps are discussed and described in detail in this work in order to better understand the development of a complex parallel CFD computation code.

Modelling The first step is the modelling of the natural phenomena and processes by defining a mathematical model. The necessary parameters have to be determined, irrelevant

parts have to be omitted, and relations have to be created and consistently formulated. The resulting model should be *as simple as possible, but not simpler*[6]. This step usually results in a system of coupled differential equations. The governing fluid equations are derived in Chapter 2.

Numerical Treatment Once a mathematical model has been established, it has to be discretised so that it can be processed by computers, as these are not capable of dealing with a real continuum, but only with discrete floating-point numbers. In the discretisation step, a spatial and temporal discretisation is chosen such as finite differences, finite volumes, or finite elements. Questions such as the choice of numerical algorithms are also dealt with here. Chapter 3 deals with the numerical treatment of the fluid flow equations.

Implementation The implementation phase addresses much more than just the pure choice of a programming language. Adequate memory and run time efficient data structures have to be designed, and special consideration has to be given to the parallel distribution of the data across different processes. It is very important that the parallel concept is integrated at very early design stages, as it has a huge impact on data structures and other design choices. Implementation aspects, data structure handling, parallel distribution, load balancing strategies, and other relevant questions are addressed in Chapter 4. After testing the implemented algorithms, it might be necessary to rethink the design decision, and to move back to the numerical or modelling section in order to change the methods. Bi-directional arrows in Figure 1.2 illustrate this moving forward and backward through the simulation design pipeline according to results from certain stages.

Visualisation Visualisation is a crucial process, as numerical simulations tend to produce a huge amount of data which can usually not be reduced to a single significant number. Thus, visual exploration of a velocity field, for example, can provide a helpful insight into the simulation results. Chapter 5 deals with conventional post-processing visualisation of simulation results such as velocity and temperature fields, and also introduces the notion of interactive visual exploration which can provide a better insight into a running simulation while avoiding large data transfers.

Verification and Validation From an engineering point of view, the overall verification and validation of generated data is of the utmost importance, as here the results are compared with known benchmark solutions generated, for example, from other validated software (verification) or from physical experiments (validation). Every single, previous step has to be verified, starting from the mathematical model generated, the numerical treatment, the implementation, and the visualisation. Chapter 6 deals with the verification and validation of the simulation pipeline established.

[6] quote attributed to Albert Einstein

Embedding Embedding in this context means integrating the piece of software generated into a larger project or software package. Interfaces have to be defined in order to interact with other components of such a larger environment. In this work, the embedding is done by creating a program library containing all CFD simulation routines and coupling them with post-processing tools for thermal simulation evaluations.

Real applications, which consist of more complex examples than the previously computed validation examples, are presented in Chapter 7. Hence, the use of the overall software concept can be demonstrated for post-processing and/or for coupling different tools to the simulation pipeline created, in order to evaluate the results of the computed fluid flow from multiple aspects, such as thermal comfort, air quality, or other engineering applications.

Chapter 8 summarises the contributions made by this thesis and concludes with an outlook of future work and possible improvements.

2. Mathematical Modelling of the Navier-Stokes Equations

The main goal of this thesis is to simulate indoor airflow scenarios, hence a few premises can be made in order to simplify the physical theory applied. It is sufficient to simulate subsonic single-phase fluid flows with $Ma < 0.3$, Ma designating the Mach number, and to describe the ratio of fluid velocity and the speed of sound in the given fluid. At these Mach numbers, the fluid can be regarded as incompressible [FP02, Hir07]. Furthermore, air as well as water can be regarded as *Newtonian fluid*, i. e. the approximation of a linear relationship between the non-isotropic parts of the stress and the rate-of-strain tensors is accurate for all but extreme conditions [Bat00].

For a detailed historical derivation of the basic governing equations describing fluid flows, the reader is referred to the standard literature such as [Bat00] or [Hir07]. For the sake of completeness, the basic equations are derived in the next section closely following the structure of Ferziger and Perić [FP02].

2.1. Conservation Principles

This section focuses on the basic conservation principles for a given control volume (CV) or a given control mass (CM). A conservation law for a given extensive property such as mass or momentum in a control volume focuses on the changes in the given quantity with respect to external forces.

For a given control mass it is well understood that mass m cannot be created or destroyed over time leading to the conservation of mass in a control mass

$$\frac{\mathrm{d}m}{\mathrm{d}t} = 0 \quad . \tag{2.1}$$

Furthermore, Newton's second law of motion

$$\frac{\mathrm{d}(m \cdot \vec{u})}{\mathrm{d}t} = \sum \vec{f} \tag{2.2}$$

for a given velocity $\vec{u} = (u_1\ u_2\ u_3)^T$ and external forces $\vec{f}$ on the control mass describes the conservation of momentum. u_1, u_2, u_3 here represent the velocity components in the three spatial dimensions x, y, z.

The control mass principle is also called the *Lagrangian formulation*, where an observer is moving with the same speed $\vec{u}$ as the observed system, which is often used in solid mechanics. In fluid mechanics, however, it is easier to focus on a stationary (control) volume which is crossed by a fluid flow, known as a *Eulerian formulation*. Now, both conservation laws (2.1) and (2.2) have to be reinterpreted in the context of control volumes. Therefore, *intensive* quantities (i. e. independent of the amount of observed matter) are used instead of *extensive* quantities. The conversion of an intensive quantity ϕ to an extensive quantity Φ can be described by

$$\Phi := \int_{V_{CM}} \rho\phi \mathrm{d}V \quad , \tag{2.3}$$

where V_{CM} is the volume of the control mass. Examples of intensive quantities are density ρ (mass per unit volume) and velocity $\vec{u}$ (momentum per unit volume). Using this definition of intensive quantities, the mass conservation can be described by using $\phi = 1$, and the momentum equation by using $\phi = \vec{u}$.

Using Reynolds' transport theorem and relation (2.3), the left-hand side of every conservation equation can be expressed using control volumes by

$$\frac{\partial}{\partial t}\int_{V_{CM}} \rho\phi \mathrm{d}V = \frac{\partial}{\partial t}\int_{V_{CV}} \rho\phi \mathrm{d}V + \oint_{S_{CV}} \rho\phi\left((\vec{u} - \vec{u}_b)\cdot\vec{n}\right)\mathrm{d}S \quad , \tag{2.4}$$

with V_{CV} representing the volume and S_{CV} the surface of the control volume. $\vec{n}$ is the unit vector perpendicular to the given control volume surface pointing outwards. $\vec{u}$ is the velocity of the fluid and $\vec{u}_b$ the velocity of the moving control volume. In the case of a non-moving control volume, $\vec{u}_b = 0$ simplifies the equation.

Expressed in words, (2.4) can be interpreted as follows: the rate of change of an extensive property Φ over time in a control mass is equal to the rate of change of the intensive property ϕ in the control volume plus the relative flux through all sides of the control volume. The surface integral in Equation (2.4) is often referred to as the convective flux of ϕ. In the special case that the control volume is moving with the same rate as the fluid, the control volume corresponds to the control mass, as $\vec{u} - \vec{u}_b = 0$ and thus the convective flux is zero.

2.2. Conservation of Mass

Using the conservation of mass (2.1), the intensive quantity $\phi = 1$, and Reynolds' theorem (2.4), the conservation of mass in integral form can be written as

$$\frac{\partial}{\partial t}\int_{V} \rho \mathrm{d}V + \oint_{S} \rho\vec{u}\cdot\vec{n}\mathrm{d}S = 0 \quad . \tag{2.5}$$

By applying Gauss' theorem [BSMM08] which states that the surface integral of a flux $\vec{\varphi}$ is equal to the volume integral of the divergence of the flux $\vec{\varphi}$, given as

$$\oint_S \vec{\varphi} \cdot \vec{n} \mathrm{d}S = \int_V \nabla \cdot \vec{\varphi} \mathrm{d}V \quad , \tag{2.6}$$

Equation (2.5) can be written as

$$\frac{\partial}{\partial t} \int_V \rho \mathrm{d}V + \int_V \nabla \cdot (\rho \vec{u}) \mathrm{d}V = 0 \quad , \tag{2.7}$$

while ensuring that the prerequisites of Gauss' theorem for a continuous flux and an existing, continuous partial derivative of $\rho\vec{u}$ are fulfilled. Since Equation (2.7) is written for an arbitrary control volume V, it has to be valid at every point, which can be expressed using the differential form of the equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \quad . \tag{2.8}$$

In the case of an incompressible, isothermal, single-phase fluid flow, the density can be assumed to be constant over space and time. Thus, the partial derivative of the density with respect to time is zero and the continuity equation can be simplified to the following expression

$$\nabla \cdot \vec{u} = 0 \quad , \tag{2.9}$$

or, using 3D Cartesian coordinates, as

$$\frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2} + \frac{\partial u_3}{\partial x_3} = 0 \quad . \tag{2.10}$$

It should be noted here again that the differential form (2.8) is more restrictive, as Gauss' theorem was applied so as to require the flux to be differentiable, i. e. having at least C1 continuity. Thus, certain problems involving shock, for example, cannot be computed using the differential form. But as the application scenarios of this work do not include shock or similar effects, the differential form can be used further on.

It should also be mentioned that a differential form is called *conservative* if and only if all spatial derivatives can be grouped under a divergence operator, otherwise they are in *non-conservative* form. If a very fine grid is chosen in the discretisation step, both forms will provide the same results. On coarse grids, however, the non-conservative formulation may introduce additional numerical errors which act as internal sources or sinks and might destroy the conservation properties, which could lead to significant discrepancies in the results [FP02, Hir07].

2.3. Conservation of Momentum

Using the conservation of momentum in a control mass (2.2) and Reynolds' theorem (2.4) with $\phi = \vec{u}$, the momentum equation can be written for a non-moving control volume V with the surface S as

$$\frac{\partial}{\partial t}\int_V \rho\vec{u}\mathrm{d}V + \oint_S \rho\vec{u}(\vec{u}\cdot\vec{n})\mathrm{d}S = \sum \vec{f} \quad . \tag{2.11}$$

In order to express $\sum \vec{f}$ with intensive quantities, all the forces acting on the control volumes have to be summed up, namely all surface forces (such as pressure, shear tensions, etc.) and also all volume forces (such as gravitation, Coriolis force, etc.).

The surface forces resulting from pressure and viscous stresses can be interpreted as microscopic momentum fluxes through the control volume surface. In order to obtain a closed system of equations where all the quantities can be expressed using density and velocity, approximations are made to the viscous model. As only Newtonian fluids are considered here, the stress tensor $\mathbf{T}$ representing the molecular transport flux can be written as

$$\mathbf{T} = -\left(p + \frac{2}{3}\mu\nabla\cdot\vec{u}\right)\mathbf{I} + 2\mu\mathbf{D} \tag{2.12}$$

with p describing the static pressure, μ the dynamic viscosity, $\mathbf{I}$ the unity tensor and $\mathbf{D}$ the tensor of the deformation rate

$$\mathbf{D} = \frac{1}{2}\left(\nabla\vec{u} + (\nabla\vec{u})^T\right) \quad . \tag{2.13}$$

If volume forces such as gravity are clustered in a vector $\vec{b}$, the momentum equations can be written as

$$\frac{\partial}{\partial t}\int_V \rho\vec{u}\mathrm{d}V + \oint_S \rho\vec{u}(\vec{u}\cdot\vec{n})\mathrm{d}S = \oint_S \mathbf{T}\cdot\vec{n}\mathrm{d}S + \int_V \rho\vec{b}\mathrm{d}V \quad , \tag{2.14}$$

with the i-th component given as

$$\frac{\partial}{\partial t}\int_V \rho u_i\mathrm{d}V + \oint_S \rho u_i(\vec{u}\cdot\vec{n})\mathrm{d}S = \oint_S \vec{t_i}\cdot\vec{n}\mathrm{d}S + \int_V \rho b_i\mathrm{d}V \quad , \tag{2.15}$$

with

$$\vec{t_i} = \mu\nabla u_i + \mu(\nabla\vec{u})^T\cdot\vec{e_i} - \left(p + \frac{2}{3}\mu\nabla\cdot\vec{u}\right)\vec{e_i} \tag{2.16}$$

using $\vec{e_i}$ as the Cartesian unity vector in the direction of the coordinate x_i. In 3D Cartesian coordinates, this relation can be written as

$$\vec{t_i} = \mu\sum_{j=1}^{3}\left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\vec{e_j}\right] - \left(p + \frac{2}{3}\mu\sum_{j=1}^{3}\frac{\partial u_j}{\partial x_j}\right)\vec{e_i} \quad . \tag{2.17}$$

By applying Gauss' theorem to the surface integrals and assuming continuity for the fluxes, the momentum equation in conservative differential form for the i-th direction can be written as

$$\frac{\partial(\rho u_i)}{\partial t} + \nabla \cdot (\rho u_i \vec{u}) = \nabla \cdot \vec{t_i} + \rho b_i \quad . \tag{2.18}$$

Introducing (2.16) into Equation (2.18) and regrouping terms yields the momentum equation for a Newtonian fluid

$$\frac{\partial(\rho u_i)}{\partial t} + \nabla \cdot (\rho u_i \vec{u}) = \nabla \cdot (\mu \nabla u_i) + \frac{1}{3}\nabla \cdot (\mu \nabla \cdot \vec{u}) - \nabla \cdot (p\vec{e_i}) + \rho b_i \quad . \tag{2.19}$$

Navier-Stokes Equations for Incompressible Newtonian Fluid Flows

In the case of incompressible flows (i. e. $\rho = \text{const}$), the continuity equation $\nabla \cdot \vec{u} = 0$ can be introduced and the equation for the i-th component can be written as

$$\frac{\partial u_i}{\partial t} + \nabla \cdot (u_i \vec{u}) = \nabla \cdot (\nu \nabla u_i) - \frac{1}{\rho}\nabla \cdot (p\vec{e_i}) + b_i \quad , \tag{2.20}$$

where $\nu = \mu/\rho$ is called the kinematic viscosity. Together with the continuity equation (2.9), Equations (2.20) are called the Navier-Stokes equations for an incompressible, isothermal Newtonian fluid flow.

It should be noted that diffusive fluxes appear as second order differential terms in Equation (2.20), whereas convective fluxes appear as a first order differential term.

Euler Equations for Inviscid Flows

Another special case is the non-viscous Euler flow. In contrast to Equation (2.12), the stress tensor reduces to $\mathbf{T} = -p\mathbf{I}$, reducing the Navier-Stokes equations to the Euler equations

$$\frac{\partial(\rho u_i)}{\partial t} + \nabla \cdot (\rho u_i \vec{u}) = -\nabla \cdot (p\vec{e_i}) + \rho b_i \quad , \tag{2.21}$$

together with the continuity equation (2.8). As the flow is non-viscous, it does not adhere to walls and no boundary layer can be resolved. It is usually used for simulating compressible fluids at high velocities. This form of the equations will not be dealt with further, as it is not relevant for the given application scenarios.

Dimensionless Quantities

The Navier-Stokes equations are sometimes normalised in order to achieve scaling to real flow scenarios. This is done by introducing flow specific constants, such as the characteristic length L_0 or the characteristic velocity v_0, and by replacing the basic variables such as space $x_i^\star = x_i/L_0$ or velocities $u_i^\star = u_i/v_0$. Furthermore, flow-specific numbers characterising the fluid are introduced, such as

- Reynolds number, characterising the ratio of inertia to viscosity

$$Re := \frac{\rho v_0 L_0}{\mu} \quad , \tag{2.22}$$

- Froude number, characterising the ratio of inertia to gravity

$$Fr := \frac{v_0}{\sqrt{L_0 g}} \quad . \tag{2.23}$$

There are many more characteristic numbers which have an influence on the fluid. For the purpose of this work, they are introduced when they are needed.

Although the dimensionless analysis of fluid flows can simplify matters with respect to scalability, it also introduces some major drawbacks. The question of a characteristic length can be answered very quickly for a standard benchmark example such as a cylinder in a channel, but in more complicated settings, this matter is not so easily settled. Therefore, the code discussed in this thesis does not use a dimensionless approach, but rather base SI units[1] or derived SI units such as pascal (Pa := kg/(m·s^2)).

In order to be able to simulate a certain benchmark scenario for a given Reynolds number, the necessary viscosity μ is computed from Equation (2.22) using given scenario parameters.

For the sake of completeness, the dimensions of the quantities used for the Navier-Stokes equations are the following: velocities u_i in [m/s], time t in [s], dynamic viscosity μ in [kg/(s·m)], kinematic viscosity ν in [m^2/s], density ρ in [kg/m^3], pressure p in [Pa], and external volume forces (such as acceleration due to gravity) b_i in [m/s^2].

2.4. Conservation of Energy

Using the previously introduced Reynolds' transport theorem, the integral form of the energy conservation for a generic intensive variable ϕ in a non-moving control volume V with surface S can be written as

$$\frac{\partial}{\partial t}\int_V \rho\phi \mathrm{d}V + \oint_S \rho\phi\vec{u}\cdot\vec{n}\mathrm{d}S = \sum f_\phi \quad , \tag{2.24}$$

[1] The International System of Units (SI): `http://www.bipm.org/en/si`

where f_ϕ characterises the transport of ϕ by all mechanisms except convection as well as all sources of the scalar property. As diffusive transport is always present (even in a motionless fluid), the generic approximation for diffusion can be introduced as

$$f_\phi^d = \oint_S \Gamma \nabla\phi \cdot \vec{n} \mathrm{d}S \quad , \tag{2.25}$$

where Γ denotes the diffusion coefficient of the intensive property ϕ. If internal volume sources are clustered to q_ϕ, the generic convection-diffusion equation for an incompressible fluid can be written in integral form as

$$\frac{\partial}{\partial t}\int_V \rho\phi \mathrm{d}V + \oint_S \rho\phi\vec{u} \cdot \vec{n} \mathrm{d}S = \oint_S \Gamma\nabla\phi \cdot \vec{n} \mathrm{d}S + \int_V q_\phi \mathrm{d}V \tag{2.26}$$

and in conservative differential form (assuming continuous fluxes) by using Gauss' theorem as

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot (\rho\phi\vec{u}) = \nabla \cdot (\Gamma\nabla\phi) + q_\phi \quad . \tag{2.27}$$

2.4.1. Thermal Computations and Simplifications

Using (2.26), it is possible to model a large number of phenomena, such as chemical concentrations, for example. This work will focus on the convection-diffusion equation for the heat transfer, however, with $\phi = c_p \cdot T$, $\Gamma = k/c_p$, and $q_\phi = q_{int}$, where c_p describes the specific heat capacity at constant pressure in [J/(kg·K)], T the temperature in [K], k the heat conduction coefficient in [W/(m·K)], and q_{int} the internal heat generation in [W/m^3]. By assuming a constant specific heat capacity c_p in the fluid, the generic mass diffusion term in Equation (2.27) can be written as

$$\nabla \cdot (\Gamma\nabla\phi) = \nabla \cdot \left(\frac{k}{c_p}\nabla(c_p T)\right) = \nabla \cdot (k\nabla T) \quad , \tag{2.28}$$

where the term in brackets on the far right actually represents a physical heat diffusion process called *Fourier's law of heat conduction*.

In order to further simplify the calculations, certain assumptions can be made:

- Variations in pressure are small enough to have no influence on the thermodynamic properties. According to Lienhard [LL11], this approximation is reasonable for most liquid flows and flowing gases at moderate velocities, namely slower than $Ma < 0.3$. As the thermal simulation should be able to compute natural as well as mixed convection phenomena for indoor airflow settings with moderate fluid velocities, it is reasonable to make this assumption.
- Changes in density occur only due to changes in temperature, and there exists a linear correlation between the two variables. This approximation is called the *Boussinesq approximation*. The density is assumed to be constant everywhere except in the part

of the momentum equations (2.20) dealing with body forces b_i such as gravity. This approximation introduces errors in the order of less than 1%, if the temperature difference is smaller than 2 K in water or 15 K in air. Further information on the validity of the Boussinesq approximation can be found in [FP02] or [GDN98].

- All other fluid properties such as thermal conductivity k or dynamic viscosity μ are independent of any temperature change and will be regarded as constant.
- Viscous dissipation in the fluid is negligible and the fluid does not heat up significantly due to this effect.

With these assumptions, Equation (2.27) for simulating a transient heat transfer involving convection, diffusion, and internal heat sources can be written in conservative differential form as

$$\frac{\partial T}{\partial t} + \nabla \cdot (T\vec{u}) - \nabla \cdot (\alpha \nabla T) - \frac{q_{int}}{\rho \cdot c_p} = 0 \quad , \tag{2.29}$$

where $\alpha = k/(\rho \cdot c_p)$ represents the heat diffusion coefficient in $[\mathrm{m}^2/\mathrm{s}]$.

By applying the chain rule for differentiating the convective part and introducing the continuity equation (2.9), it can be rewritten as

$$\nabla \cdot (T\vec{u}) = \nabla T \cdot \vec{u} + T \underbrace{(\nabla \cdot \vec{u})}_{=0} = \vec{u} \cdot \nabla T \quad . \tag{2.30}$$

Furthermore, by assuming α is constant, (2.29) can be written in the well-known, although non-conservative form

$$\frac{\partial T}{\partial t} + \vec{u} \cdot \nabla T - \alpha \Delta T - \frac{q_{int}}{\rho \cdot c_p} = 0 \quad . \tag{2.31}$$

2.4.2. Boussinesq Approximation

Equation (2.31) describes convective and diffusive processes for the heat transfer in the computational domain. These changes again have an influence on the velocity field, however. Hence, the resulting temperature field needs to be coupled to the momentum equations. This back-coupling can be quite complex as the temperature-dependent density is present in every term of the momentum equations (2.19). By applying the Boussinesq approximation, the density is still assumed to be constant (marked as ρ_∞) in the transient and convective term and only varies in the body force term b_i (which accounts for gravity)

$$\frac{\partial \rho_\infty u_i}{\partial t} + \nabla \cdot (\rho_\infty u_i \vec{u}) = \nabla \cdot (\mu \nabla u_i) - \nabla \cdot (p\vec{e}_i) + \rho(T) b_i \quad . \tag{2.32}$$

Furthermore, a linear relation between temperature and density is assumed

$$\rho(T) = -\rho_\infty \cdot \beta \cdot (T - T_\infty) \quad , \tag{2.33}$$

where β represents the thermal expansion coefficient of the fluid in [1/K], and T_∞ the temperature of the undisturbed fluid at rest in [K]. For an ideal gas the coefficient of thermal expansion can be computed by $\beta = 1/T_\infty$, for other materials it can be taken from tabulated values, such as Bejan [Bej13].

Thus, the modified momentum equations using the Boussinesq approximation can be written as

$$\frac{\partial \rho_\infty u_i}{\partial t} + \nabla \cdot (\rho_\infty u_i \vec{u}) = \nabla \cdot (\mu \nabla u_i) - \nabla \cdot (p\vec{e_i}) - \rho_\infty \cdot \beta \cdot (T - T_\infty) g_i \quad , \tag{2.34}$$

where g_i is the gravity force in the direction i in [m/s^2].

It is obvious that, in the case of an isothermal fluid at T_∞, the buoyancy term (i. e. the last term in (2.34)) cancels out and no effects can be seen. As soon as one wall is heated, however, the buoyancy terms start to grow and will set the fluid in motion.

It should also be noted that the linear assumption (2.33) can be wrong, even if the temperature differences are small enough. One example is water at around 4°C, as the temperature-density relation is non-linear and the density has a maximal value around this point [FP02].

Some more characteristic fluid numbers should be introduced at this point, as they are responsible for fluid flow properties of natural and mixed convection phenomena:

- Prandtl number describing the ratio of momentum diffusivity to thermal diffusivity

$$Pr := \frac{\nu}{\alpha} \quad , \tag{2.35}$$

- Grashof number describing the ratio of buoyancy to viscous forces acting on a fluid

$$Gr := \frac{|\vec{g}|\beta(T_{hot} - T_{cold})L_0^3}{\nu^2} \quad , \tag{2.36}$$

  where L_0 is the characteristic length scale of the system, and T_{hot} and T_{cold} characteristic temperatures in the system,

- Rayleigh number

$$Ra := Pr \cdot Gr \quad . \tag{2.37}$$

  If $Ra < Ra_{crit}$ the heat transfer process is dominated by conduction effects, whereas if $Ra > Ra_{crit}$ the heat transfer is dominated by convection effects.

- Richardson number describing the ratio of natural convection to forced convection

$$Ri := \frac{|\vec{g}|\beta(T_{hot} - T_{cold})L_0}{v_0^2} = \frac{Ra}{Re^2} \quad , \tag{2.38}$$

  where v_0 denotes the characteristic velocity and Re the Reynolds number.

2.5. Boundary Conditions

In order to completely define the mathematical model, boundary conditions have to be applied. As a matter of notation, u_n and x_n depict the velocity component and the coordinate axis *normal* to the boundary condition, respectively. u_t and x_t represent the velocity component and the coordinate axis *tangential* to the given boundary, respectively.

In accordance with [Bej13], [Hir07], and [GDN98] the following boundary conditions can be applied for the velocity components of viscous, incompressible fluid flows.

- No-slip boundary conditions at walls: the tangential velocity component u_t of the flow has the same velocity as the wall itself (Dirichlet condition), and the normal velocity component u_n must be zero (non-penetrability of walls). In the case of a non-moving wall, this means that the fluid in the immediate vicinity of the wall must be at rest ($u_n = 0,\ u_t = 0$). This is the most frequently applied boundary condition representing a real physical wall.

- Slip boundary conditions (also referred to as *free-slip* boundary condition): a slip boundary condition is mathematically identical to a symmetry boundary condition. Physically, it means that the tangential velocity component u_t does not stick to the wall. Therefore, the partial derivative of the tangential component $\partial u_t/\partial x_n$ has to be zero (homogeneous Neumann condition). Furthermore, the normal component u_n of the velocity must be zero, as the wall is non-penetrable for the fluid.

- Inflow boundary conditions: there are different kinds of possible inflow boundary conditions which have to be set with care in order to be compatible with other given conditions in the setup. In the following, explicitly prescribed velocities u_i are applied, having a given magnitude and direction vector (Dirichlet conditions). Another possibility would be to prescribe the pressure and compute the inflow velocity resulting from a driving pressure gradient.

- Outflow boundary conditions: as in the case of the inflow conditions, a different set of outflow conditions can be used. In the following the outflow conditions are set in such a way that the total velocity does not change in the direction normal to the boundary. This can be achieved by setting the normal derivatives of the velocities to zero (homogeneous Neumann condition): $\partial u_i/\partial x_i = 0$ for all directions i. In this case, special care has to be taken in respect of the pressure boundary condition described in the next paragraph.

For the pressure values, the following boundary conditions can be set.

- Dirichlet conditions: a prescribed, fixed pressure p can be used as the boundary condition. This could be the absolute atmospheric pressure or a pressure value of zero, meaning that a pressure correction relative to zero is computed throughout the domain.

Depending on the setting of the inflow and outflow boundary conditions, Dirichlet conditions can be used. In the case of the above-described velocity boundary conditions, outflow conditions have to be Dirichlet conditions.

- Homogeneous Neumann conditions: the partial derivative of the pressure in the normal direction $\partial p/\partial x_n$ has to be zero. This boundary condition is chosen for all other boundaries mentioned above, with the exception of the outflow boundary condition.

The temperature boundary conditions can be described by the following settings.

- Fixed temperature: fixed temperature conditions are Dirichlet boundary conditions where T is imposed on the boundary. The velocity and pressure boundary conditions can be any of the above conditions.
- Heat flux condition: a prescribed heat flux q_i^{side} in [W/m^2] through the corresponding side i can be applied by using Fourier's law

$$- k \left. \frac{\partial T}{\partial x_i} \right|_{x_i=\text{boundary}} = q_i^{side} \quad , \tag{2.39}$$

  with k representing the heat conduction coefficient. This condition can be used if the heat flux traversing a boundary is known. This is the case if the heat flux through a wall was computed using another software, for example, and should be imposed on a flow enclosed by the aforementioned walls.
- Adiabatic condition: this condition is a special case of the heat flux condition, where the directional flux $q_i^{side} \equiv 0$. This condition represents a standard homogeneous Neumann condition where $\partial T/\partial x_i = 0$. Adiabatic, from the Greek word *adiabatos*, meaning *impassable*[2], defines that no heat is exchanged, thus perfect wall insulation is assumed. The wall surface temperature is adapted in order to model the zero-heat flux condition.

As a last special case for boundary conditions, *periodic* boundary conditions are introduced. Periodic boundary conditions are global boundary conditions applied to a certain direction i in the domain, where values on opposite sides of the given direction i coincide. Hence, the value from the bottom, for example, is used at the top as the boundary condition.

Further implementation details of these and other boundary conditions are given in the next chapters.

2.6. Turbulent Flow

Most of the flows which are of particular interest for engineers are turbulent in nature, and therefore need a different treatment than the laminar viscous flow considered up to now.

[2]Source: Encyclopaedia Britannica, `http://www.britannica.com`

Over the years, different approaches have been followed in order to simulate turbulent flows. The main characteristics of a turbulent flow are its highly transient, irregular, mixing, three-dimensional behaviour [FP02]. Due to the turbulent eddies in the fluid, the velocity components are subject to viscous internal friction, and kinetic energy dissipates into the fluid. It should be kept in mind that when coupling the thermal energy simulation introduced in Section 2.4.1 one of the prerequisites was that the increase in thermal energy from dissipation due to viscous friction does not heat up the fluid to any great extent.

According to [BFR80], six categories of turbulence modelling can be distinguished. Here, only the three most important ones will be discussed, a complete and extensive study can be found in [Wil94], [FK95], or [CJ97], for example. The approaches introduced and discussed here are the Reynolds-Averaged Navier-Stokes (RANS) model, the Large-Eddy-Simulation (LES), and the Direct Numerical Simulation (DNS).

2.6.1. Reynolds-Averaged Navier-Stokes (RANS)

The RANS equations are averaged equations first introduced by O. Reynolds in 1895 [Rey95]. The basic idea is that every variable ϕ can be expressed as the sum of an averaged value $\overline{\phi}$ and a variation ϕ' as

$$\phi(\vec{x},t) = \overline{\phi}(\vec{x}) + \phi'(\vec{x},t) \quad . \tag{2.40}$$

Depending on the flow characteristics, three different averaging techniques can be applied. In a statistically steady flow, a *time average* can be computed by

$$\overline{\phi}(\vec{x}) = \lim_{T\to\infty} \frac{1}{T} \int_0^T \phi(\vec{x},t)\mathrm{d}t \quad , \tag{2.41}$$

where t represents the time, T the averaging interval over the time and $\vec{x}$ the spatial coordinate of the variable. The averaging interval T has to be bigger than the time scale of the fluctuations in order to separate the turbulent effects.

In the case of a homogeneous turbulence, i. e. a turbulent flow which is uniform in all directions on average, a *spatial average* can be expressed. The most general type of averaging is the *ensemble averaging* which can be applied to any problem. N identical experiments have to be performed, where the average is then obtained by

$$\overline{\phi}(\vec{x},t) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \phi_n(\vec{x},t) \quad , \tag{2.42}$$

where N has to be big enough to eliminate any turbulent fluctuations. As most engineering-based turbulence problems are inhomogeneous, (2.42) has to be applied most of the time.

Using Equation (2.41), it can be shown that $\overline{\phi'} = 0$. Linear terms in the conservation equation will yield only one term as the averaged value, whereas non-linear terms will create

additional terms, so-called correlation effects. A quadratic term produces two terms, namely a product of the mean values and a correlation of fluctuations

$$\overline{u_i \cdot \phi} = \overline{(\overline{u_i} + u_i') \cdot (\overline{\phi} + \phi')} = \overline{u_i} \cdot \overline{\phi} + \overline{u_i' \cdot \phi'} \quad . \tag{2.43}$$

The last term vanishes only for uncorrelated variables, which is not the case in the conservation equations. Thus, Reynolds-averaged equations will always contain terms such as $\rho \cdot \overline{u_i' \cdot u_j'}$ called *Reynolds stress* and $\rho \cdot \overline{u_i' \cdot \phi'}$ called *turbulent scalar flux*, where ϕ is an arbitrary scalar quantity.

Using these notations and definitions, the incompressible Navier-Stokes equations with the continuity equation (2.9) and the momentum equation for the i-th component (2.20) can be rewritten in Cartesian coordinates into the RANS form as

$$\sum_{j=1}^{3} \frac{\partial \overline{u_j}}{\partial x_j} = 0 \quad , \tag{2.44}$$

$$\frac{\partial(\rho\overline{u_i})}{\partial t} + \sum_{j=1}^{3} \frac{\partial}{\partial x_j} \left(\rho \cdot \overline{u_j} \cdot \overline{u_i} + \rho \cdot \overline{u_j' \cdot u_i'} \right) = -\frac{\partial \overline{p}}{\partial x_i} + \sum_{j=1}^{3} \frac{\partial}{\partial x_j} \left[\mu \cdot \left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i} \right) \right] + \rho \overline{b_i} \, . \tag{2.45}$$

Due to the Reynolds stresses occurring in the convective part and the turbulent scalar fluxes in the energy equation, the RANS equations are no longer closed. This is referred to as the *closure problem* in general literature [Wil94].

In order to tackle this problem, approximations have to be applied by means of empirical parameters or parameter equations for the Reynolds stresses and the turbulent scalar fluxes. Different models can be taken into account, ranging from a simple algebraic model (also called zero-equation model) as proposed by [vK30], one-equation models of turbulence postulated by [Pra45] through to a two-equation model proposed by [Kol42] formulating the k-ω-model, where k models the kinetic energy of the turbulent fluctuations and ω the rate of energy dissipation in unit volume and time. Due to the lack of computing power at the beginning of the 20th century, the model did not receive much attention. With the arrival of modern computers and considerably more computing power, this approach was further investigated, and Launder and Spalding [LS72] introduced their k-ε-model, which is so far the most commonly used model using the RANS approach, even though the model is demonstrably inadequate for flows with adverse pressure gradients [RS86].

For a very good and detailed discussion about advantages and disadvantages of the different models, as well as their detailed numerical forms and parameters, the interested reader is referred to [Wil94].

2.6.2. Large-Eddy-Simulation (LES)

The turbulent behaviour of a fluid flow is usually spread over a wide range of length and time scales. In the RANS approach, this is not taken into account. In the LES, however, a multi-scale approach is used, where larger turbulent structures (so-called *eddies*) are resolved directly, whereas smaller turbulent effects – which can usually not be captured by the chosen grid resolution – are modelled indirectly, similar to the RANS case. As movements on the large scale carry more kinetic energy than small-scale movements, a detailed analysis of large scale effects will enhance the accuracy of the overall simulation compared to a RANS approach.

In order to do a LES simulation, it is important to identify the large scale which should be resolved, and the sub-grid scale (SGS) which should be modelled. In the work of [Leo75], a filtering of the velocity field is proposed, where small-scale effects are 'cut-off'. In 3D, such a filtering operation can be written as

$$\overline{\phi}(\vec{x},t) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \phi(\vec{\xi},t')G(\vec{x}-\vec{\xi},t-t')\,\mathrm{d}t'\,\mathrm{d}^3\vec{\xi} \quad , \tag{2.46}$$

where G is a convolution kernel associated with the cut-off scales in space $\overline{\Delta}$ and time $\overline{\tau_c}$, $\vec{\xi}$ a spatial vector, and t' a temporal variable used in the convolution [Sag06]. It can be shown that the filter G has to have certain properties in order to be used together with the Navier-Stokes equations for a LES approach, such as conservation of constants, linearity, and commutation with derivation (see [Sag06] for instance). Conventional filters involve the *box (or top-hat) filter*, the *Gaussian filter*, or the *spectral (or sharp) cut-off filter*. A very detailed analysis of the filtering properties can be found in Chapter 2 of [Sag06]. In general, it can be stated that eddy structures with a characteristic length larger than $\overline{\Delta}$ are resolved, and eddies smaller than $\overline{\Delta}$ have to be modelled by a virtual eddy viscosity (or sub-grid viscosity). It should be noted at this point that the chosen cut-off scale in space $\overline{\Delta}$ does not necessarily need to correspond with the chosen grid discretisation h. Only the condition $\overline{\Delta} > h$ must be fulfilled when choosing either a filter size or the grid discretisation, as structures smaller than the chosen grid resolution obviously cannot be resolved directly.

As the basis of the filtering is now defined, the Navier-Stokes equations can be expressed in terms of the filtered values, analogously to the RANS equations. They will not be stated here, but the reader is advised to refer to general literature, such as [Sag06] or [FP02] for example.

The history of LES, and thus sub-scale modelling, began with Smagorinsky [Sma63], a meteorologist who carried out climatological research involving different scales. *It is interesting to note that Smagorinsky's model was a total failure as far as the atmosphere and oceans are concerned, because it dissipates the large-scale motions too much. It was an immense success, though, with users interested in industrial-flow applications, which shows that the outcomes of research are as unpredictable as turbulence itself.*[3]

[3] taken from the foreword to the first edition of [Sag06], pp. XIIV-XIV, written by Marcel Lesieur

Since then, a lot of different models for the sub-grid scale have been introduced in order to describe a sub-grid viscosity. Smagorinsky's model is an *eddy viscosity model* based on the observation that the main contributions to the sub-grid scale Reynolds stress are transport and dissipation, similar to viscous effects in laminar flows. Hence, an eddy viscosity μ_t can be defined based on

$$\mu_t = C_S^2 \rho \overline{\Delta}^2 |\overline{S}| \quad , \tag{2.47}$$

where C_S is a model parameter to be defined, $\overline{\Delta}$ the cut-off scale (or filter length), and

$$|\overline{S}| = \sqrt{\overline{S_{ij}} \cdot \overline{S_{ij}}} = \sqrt{\frac{1}{2}\left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i}\right) \cdot \frac{1}{2}\left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i}\right)} \quad . \tag{2.48}$$

This form of eddy viscosity can be derived by many methods and nearly all arrive at the conclusion of $C_S \approx 0.2$, although it cannot be assumed to be a universal constant. It can take different values for different domain and/or fluid settings. The Smagorinsky model is quite simple and successful, although it needs some modifications for some types of flow, such as channel flows, where different values of C_S have to be used close to boundaries (see [MFR80], [BFR80], or [YO86]).

2.6.3. Direct Numerical Simulation (DNS)

While RANS models the turbulence completely, LES models part of the turbulence and resolves large eddy structures directly. A direct numerical simulation (DNS) is at the other end of the possible spectrum, as no part of the turbulence is modelled, but all features are completely resolved using the 3D Navier-Stokes equations. Thus, no errors due to modelling or approximations are introduced (other than the errors introduced due to the grid discretisation itself). Hence, DNS is basically the easiest way to simulate turbulence, as no special treatment has to be applied.

Unfortunately, *there ain't no such thing as a free lunch*[4], meaning that the simplicity of the simulation concept involving turbulence has to be bought by memory and computing power requirements.

Using a direct numerical simulation, the simulation domain must have at least the size of the largest turbulent structure which is of interest in order to ensure the capturing of all significant large scale effects. A measure can be the integral scale L which basically represents the distance over which the fluctuations of the velocity components are still correlated. Thus, the domain size should be a multiple of L. On the other hand, the dissipation of kinetic energy has to be represented correctly, which has to be ensured on the smallest scale where viscous effects are dominant. Hence, the grid discretisation must not be larger than the

[4]Safire, W.: 'On Language; Words Left Out in the Cold', New York Times, 14. Feb. 1993
http://www.nytimes.com/1993/02/14/magazine/on-language-words-out-in-the-cold.html

so-called Kolmogorov length-scale

$$\eta_K := \left(\frac{\nu^3}{\varepsilon}\right)^{1/4} \quad , \tag{2.49}$$

where ν is the kinematic viscosity of the fluid and ε is the average rate of dissipation of turbulent kinetic energy per unit mass [Sag06].

For a homogeneous, isotropic turbulence and using a uniform grid distribution, every direction must have at least L/η_K grid points, which leads to quite large simulation domains. [TL72] show that this ratio is proportional to $Re_L^{3/4}$, which is a Reynolds number based on the microscopic velocity fluctuations and the integral scale L. Re_L is usually in the range of a hundredth of the macroscopic Reynolds number introduced in Equation (2.22). As the maximal time step is coupled to the minimal length scale via the CFL condition (derived in detail in Section 3.7 of this work), it can be shown that the overall computation effort is proportional to Re_L^3 [FP02].

This shows that only smaller global Reynolds numbers can be computed with a reasonable amount of computing power. Usually, DNS is used to gain a better understanding of turbulent effects or heat transfers in small domains, as well as for tuning and optimising turbulence models used in LES and RANS simulations.

2.6.4. Considerations for the Code Implemented

In the code presented in the further course of this work, no explicit modelling of turbulence has yet been implemented. On the other hand, it is obvious that turbulent flow behaviour is desired in most of the larger application cases computed for this thesis.

As the main focus of this work is the numerical treatment and implementation of a data structure for efficient parallel distribution and computation, the method of turbulence desired here is a direct numerical simulation. Thus, if the computed domain were large enough in terms of geometric size to accurately represent the integral scale L, and also possessed a fine resolution in the order of the Kolmogorov length-scale η_K, it would be capable of capturing all turbulent effects accurately. As a massive parallel computation is desired in any case, the assumption holds that if the models are refined 'a bit more', the DNS simulation would be accurate.

Nevertheless, future work should concentrate on implementing a LES-based system in order to model the (still) missing sub-grid scale turbulent behaviour in the case of a too coarse model.

2.7. Special Considerations for the Incompressible Navier-Stokes Equations

Having established the basic momentum equations (2.20) and continuity equation (2.9) for an incompressible Newtonian fluid flow, the next part, namely the numerical treatment, can be approached. Unfortunately, one mathematical problem is still unsolved: analysing Equations (2.20) and (2.9) reveals that there is no independent equation for the pressure p. The momentum equations include the pressure gradient ∇p, whereas the continuity equation does not contain p at all, which complicates the solution of the incompressible Navier-Stokes equations.

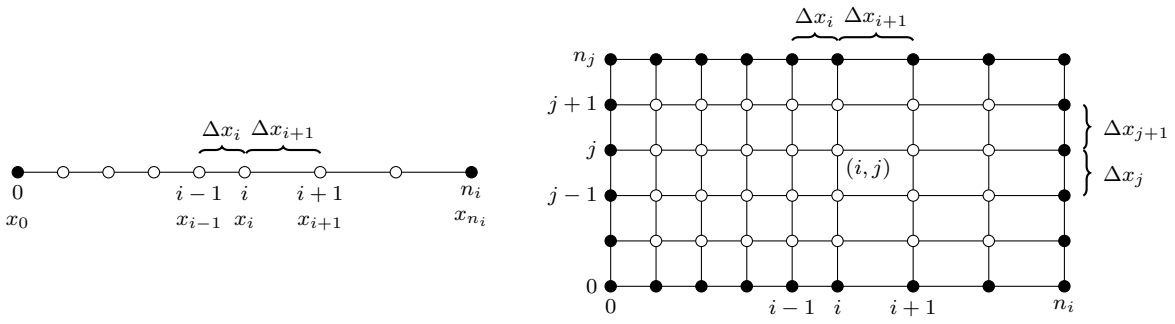
The most commonly used solutions for simulating incompressible flows are so-called *pressure correction methods*. The method was originally introduced by Harlow and Welch [HW65] in the Marker and Cell method (MAC) for the solution of free surface flows. It is very closely related to the fractional step method (or projection method) introduced by Chorin [Cho67]. A lot of modifications, alternatives, and variations have been presented in recent decades. The general idea of the applied pressure correction method is based on an iteration between velocity and pressure fields, where the pressure field acts as a correction in order to fulfil the continuity equation for the velocity field. As only the pressure gradient enters into the equations, the absolute value of the pressure is not relevant. Therefore, the pressure term p in the incompressible equations is often referred to as 'pseudo pressure' or 'working pressure'.

Numerical treatment and implementation aspects of the method used are discussed in detail in the respective Chapters 3 and 4.

3. Numerical Treatment of the Navier-Stokes Equations

The numerical treatment forms the bridge between the continuous mathematical formulations presented in Chapter 2 and the discrete world that computers can represent. This discretisation can be done using a lot of well-established methods, such as the finite difference method (FDM), the finite volume method (FVM), or the finite element method (FEM). This thesis focuses on the finite difference and finite volume methods. Details on the finite element method and its applications can be found in [ZTZ05], for example.

3.1. Finite Difference Method



(a) 1D grid setup

(b) 2D grid setup

Figure 3.1.: Non-equidistant finite difference grids.

In the finite difference approach, the computational domain is discretised using a grid. In a 1D scenario, an exemplary non-uniform grid can be seen in Figure 3.1(a). Every node can be identified by a unique integer index i and corresponds to one coordinate of the continuum x_i. A neighbouring point can be reached by increasing (right-hand neighbour) or decreasing (left-hand neighbour) the corresponding index i. In the case of a uniform grid distribution, i. e. all the distances between the nodes Δx are identical, the correlation $x_i = x_0 + i \cdot \Delta x$ can be used for computing the coordinate of the point in $\mathbb{R}$.

In a 2D scenario, an exemplary non-uniform grid can be seen in Figure 3.1(b). Analogously to the 1D case, a grid point in $\mathbb{R}^2$ can now be uniquely identified by two indices (i, j). Such a node has now four neighbours which can be reached by increasing i (east neighbour) or

j (north neighbour), or decreasing i (west neighbour) or j (south neighbour). The same principle holds for the 3D case, where three indices (i, j, k) uniquely identify a grid point in $\mathbb{R}^3$. The neighbours in the k-direction are called top $(k+1)$ and bottom $(k-1)$ neighbour.

Values other than the ones at intersections of grid lines, which are marked by circles in Figure 3.1 (used for visualisation purposes, for example), can be computed by interpolating neighbouring values. The most commonly used form of interpolation is linear interpolation (see also Section 3.2.3). In 1D, the immediate neighbours on the left- and right-hand side are linearly connected and the values interpolated along this polynomial are of first order. Higher order interpolations can be applied if more neighbouring values are involved.

Generally, in the finite difference method, partial differential equations (PDE) (such as the transient heat transfer equation (2.29) in differential form, for example) are used on a given grid. The value at each grid point represents an unknown variable and the solution of this system of unknowns forms the solution of the PDE. Thus, algebraic equations have to be set up connecting grid points with neighbouring grid points in order to represent a given PDE. These relations can be obtained by replacing differential terms from the equations by finite difference approximations on the grid. The number of equations has to coincide with the number of unknowns, as otherwise the system cannot be solved. Boundary conditions are introduced to represent boundary values (marked by black circles in Figure 3.1). If the values are given (Dirichlet conditions) no equations have to be introduced. In the case of Neumann conditions, the derivative given has to be discretised and forms an algebraic equation.

3.1.1. Approximation of First-Order Derivatives

Functions in 1D

In order to obtain a detailed formulation of the approximation terms used in the finite difference approach, the definition of the derivative of a differentiable function f shall be recalled

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \quad . \tag{3.1}$$

The discretisation of the continuous differential operator $\mathrm{d}f/\mathrm{d}x$ at a grid point x_i can be written in its discrete form using the right-hand neighbour $x_{i+1} = x_i + \Delta x$

$$\left(\frac{\mathrm{d}f}{\mathrm{d}x}\right)_i^{\text{forward}} = \frac{f(x_{i+1}) - f(x_i)}{\Delta x} \quad , \tag{3.2}$$

which represents the so-called *forward difference* at grid position i, as the right-hand neighbour x_{i+1} was used. Similarly, the left-hand neighbour $x_{i-1} = x_i - \Delta x$ could be used, leading to

$$\left(\frac{\mathrm{d}f}{\mathrm{d}x}\right)_i^{\text{backward}} = \frac{f(x_i) - f(x_{i-1})}{\Delta x} \quad , \tag{3.3}$$

which represents the *backward difference* at the grid position i. It is clear that by refining the grid and reducing Δx, the inaccuracy introduced by the approximation is reduced. By a Taylor series expansion, it can be shown that, in fact, the truncation error of the forward and backward differences is of the order $\mathcal{O}(\Delta x)$, given that f is sufficiently smooth. Thus, by reducing the step size by two, the approximation error is also reduced by a factor of two.

Another possibility for discretising the continuous differential operator is to use the *central difference* approximation. Using the notation of Figure 3.1(a), the left-hand neighbour $x_{i-1} = x_i - \Delta x_i$ and the right-hand neighbour $x_{i+1} = x_i + \Delta x_{i+1}$ are used, and the discretised difference operator at grid position i can be written as

$$\left(\frac{\mathrm{d}f}{\mathrm{d}x}\right)_i^{\text{central}} = \frac{f(x_{i+1}) - f(x_{i-1})}{\Delta x_i + \Delta x_{i+1}} \quad . \tag{3.4}$$

Using a Taylor series expansion, it can be shown that the error of the central difference approximation is of the order $\mathcal{O}\left((\Delta x)^2\right)$, given that f is sufficiently smooth.

Depending on the problem that has to be solved, approximations of higher order can be used. Polynomials and splines can be applied for interpolations of higher order, and approximations of the first derivative can be generated. As an example, an approximation of a central difference of order $\mathcal{O}\left((\Delta x)^4\right)$ on a regular grid can be written as

$$\left(\frac{\mathrm{d}f}{\mathrm{d}x}\right)_i^{\text{central4}} = \frac{-f(x_{i+2}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2})}{12\Delta x} \quad . \tag{3.5}$$

Further details and approximations can be found in [FP02].

Functions in 2D and 3D

In two and three dimensions, the discrete derivative terms can be produced in a similar way to those in the 1D case. In order to prepare the computations of discrete values at grid points, the function value f at grid position (x_i, x_j, x_k) in 3D or (x_i, x_j) in 2D will be denoted as $f_{i,j,k}$ and $f_{i,j}$ in 2D, respectively. Neighbouring values can be used by incrementing or decrementing one of the directions i, j, k. Thus, the discrete central difference operator in direction i at the position i, j, k can be written for an equidistant grid as

$$\left(\frac{\partial f}{\partial x_i}\right)_{i,j,k}^{\text{central}} = \frac{f_{i+1,j,k} - f_{i-1,j,k}}{2\Delta x_i} \quad . \tag{3.6}$$

3.1.2. Approximation of Second-Order Derivatives

Second order derivatives can be approximated in the same way as before by using first order approximations. Here a wide variety of possibilities can be combined, and the selection strongly depends on the kind of problems that should be solved. One possibility would be to

approximate everything using central differences. Thus, on an equidistant grid, the second derivative in discretised terms can be written as

$$\left(\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\right)_i^{\text{central}} = \frac{1}{\Delta x}\left[\left(\frac{\mathrm{d}f}{\mathrm{d}x}\right)_{i+\frac{1}{2}}^{\text{central}} - \left(\frac{\mathrm{d}f}{\mathrm{d}x}\right)_{i-\frac{1}{2}}^{\text{central}}\right] = \frac{f_{i+1} - 2f_i + f_{i-1}}{(\Delta x)^2} \quad . \tag{3.7}$$

For equidistant grids, it can be shown using a Taylor series expansion that the discrete second derivative (3.7) is formally of order $\mathcal{O}\left((\Delta x)^2\right)$ [FP02]. Approximations of higher order can be constructed using multiple data points in the same way as for the first derivative. For example, a central approximation of order $\mathcal{O}\left((\Delta x)^4\right)$ can be defined as

$$\left(\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\right)_i^{\text{central4}} = \frac{-f_{i+2} + 16f_{i+1} - 30f_i + 16f_{i-1} - f_{i-2}}{12(\Delta x)^2} \quad . \tag{3.8}$$

Partial differential terms can be handled exactly as in the case of the first-order derivative. In most cases, central differences for the outer and the inner derivatives are used. The term 'outer' and 'inner' derivative designates the place where the chosen discretisation is applied. In Equation (3.7), the outer derivative is a first-order central difference which uses derivatives of the function f as values. These two derivatives are called inner derivatives as they are arguments to the outer one. Depending on the physical problem, some other discretisation might be advantageous such as 'upwind' differences, etc.

3.2. Finite Volume Method

In a finite volume method, the integral equations are used as the basis, such as (2.26) for a generic convection-diffusion equation. The computational domain is divided into a finite number of control volumes bounded by volume faces. A conservation law has to hold for each and every control volume, as well as for the complete domain. Recalling Equation (2.26) and removing any internal sources q_ϕ leads to

$$\frac{\partial}{\partial t}\int_V \rho\phi \mathrm{d}V + \oint_S \rho\phi\vec{u}\cdot\vec{n}\mathrm{d}S = \oint_S \Gamma\nabla\phi\cdot\vec{n}\mathrm{d}S \quad . \tag{3.9}$$

It can be easily seen that the internal surface integrals in-between two adjacent control volumes cancel out and only the outer surfaces remain. This mathematical property is used for enforcing the physical global conservativeness of the system and is one of the great advantages of the finite volume method.

Figure 3.2 shows a 2D finite volume grid with typical naming conventions. A control volume (often referred to as a 'cell' in the following text) is bounded in 2D by four sides named after the compass rose ('e' east for positive x-direction, 'w' west for negative x-direction, 'n' north for positive y-direction, and 's' south for negative y-direction). In 3D, two additional sides are added: the directions are extended by 't' top for positive z-direction and 'b' bottom for negative z-direction. Combinations of these directions can be used to characterise a corner in

2D or an edge in 3D ('nw' north-west, for example) and a corner in 3D ('tse' top-south-east, for example). Furthermore, in a local concept, the cell 'P' shares its north side with the neighbouring cell 'N', whereas in a global concept, this cell can be referenced by $(i, j+1, k)$ index coordinates, analogously to the grid in the finite difference method. Please note that small letters denote a cell face or corner, whereas capital letters describe the cell centre.

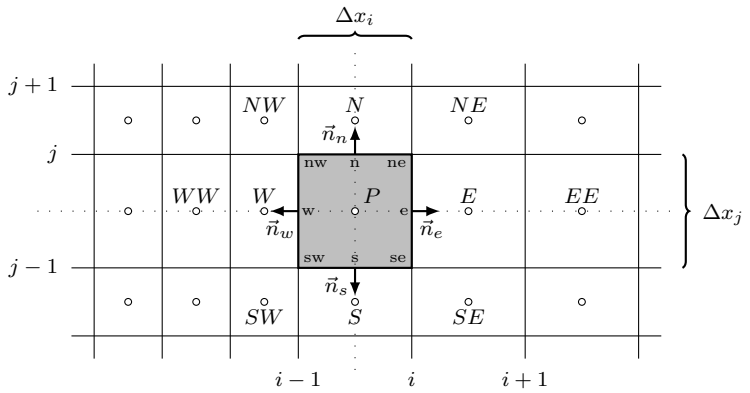


Figure 3.2.: Non-equidistant 2D finite volume grid with descriptions of directions and vectors normal to surfaces.

In order to obtain an algebraic system of equations, the surface and volume integrals have to be approximated by means of numeric integration. If values at certain points are not available, they have to be interpolated. The basic integration and interpolation techniques for a finite volume method are highlighted in the next sections by analysing the generic convection-diffusion equation.

3.2.1. Approximation of Surface Integrals

The surface integrals can be discretised for an orthogonal grid by summing up the surface integrals of the four or six sides in 2D and 3D, respectively, i. e.

$$\oint_S f \mathrm{d}S = \sum_{k=1}^{n_{sides}} \oint_{S_k} f \mathrm{d}S \quad , \tag{3.10}$$

where f denotes the content of the integrals of Equation (2.26), for example, and S the surfaces of the control volume. In order to exactly compute the result of the surface integral, the values of f would be needed on the complete surface S_k. Unfortunately, the values are usually only known at certain points in the control volume such as the centre. Thus, certain approximations have to be introduced: the integral over a face of the control volume is approximated at certain points (called integration points) only. The values at the integration points are usually approximated as functions of given node values by interpolation. Derivatives of variables at integration points are approximated using finite differences.

The easiest approximation is given by the mid-point rule and has an accuracy of second order if the value of f is known at the given side i defined by

$$\oint_{S_i} f \mathrm{d}S \approx f_i S_i \tag{3.11}$$

where S_i is the area of the side i bounding the control volume. In 2D, a second possibility would be to use the trapezoidal rule and approximate the integral of the east side, for example, by

$$\oint_{S_e} f \mathrm{d}S \approx \frac{1}{2} S_e \cdot (f_{ne} + f_{se}) \quad , \tag{3.12}$$

using the notations introduced in Figure 3.2. As in the case of finite differences, approximations of higher order can be constructed. For the east side in 2D, for example, an approximation of 4th order using the Simpson rule can be written as

$$\oint_{S_e} f \mathrm{d}S \approx \frac{1}{6} S_e \cdot (f_{ne} + 4 f_e + f_{se}) \quad . \tag{3.13}$$

To conserve the order of accuracy of the integration, unknown values have to be approximated using an interpolation technique providing at least the same order of accuracy. Compatible interpolation methods are dealt with in Section 3.2.3.

3D integration rules of higher order can be constructed analogously to Equation (3.12) or (3.13), but they tend to get complicated and difficult to implement. Therefore, in standard finite volume methods, the midpoint rule is typically used (here in 3D) and is given by

$$\oint_S f \mathrm{d}S \approx \sum_{i=1}^{6} f_i S_i = f_e S_e + f_w S_w + f_n S_n + f_s S_s + f_t S_t + f_b S_b \quad . \tag{3.14}$$

3.2.2. Approximation of Volume Integrals

Volume integrals can be treated similarly to surface integrals but without the need to sum over all sides. Instead, the midpoint rule can be used again to produce an approximation

$$\int_V q \mathrm{d}V \approx q_P \Delta V \quad , \tag{3.15}$$

where q_P is the value of the variable q at the centre 'P' of the control volume and ΔV the volume occupied by the control volume. Usually, the values q are known in the centre of the control volume and, thus, no interpolation is necessary. If q is a constant or linear function, it can be shown that the approximation is exact. For all other continuously differentiable functions, the approximation has an accuracy of second order.

Rules for methods of higher order can be constructed provided that the interpolation techniques for computing approximations at positions with unknown values have the same order

of accuracy. Further details can be found in [FP02], for example.

3.2.3. Interpolation Methods

Interpolation methods have to be used if values of variables are needed at positions where they are not stored or not available yet. The generic convection-diffusion equation (2.26) contains two surface integrals: the convective part $f^c = \rho\phi\vec{u} \cdot \vec{n}$ and the diffusive part $f^d = \Gamma\nabla\phi \cdot \vec{n}$. All the necessary variables such as velocities $\vec{u}$, density ρ, etc. have to be interpolated at the control volume boundaries from known locations, such as the cell centres.

This can be done with a method of a certain order of accuracy. The interpolation methods described use the east side 'e' by way of example. They are, however, valid for all other sides if the corresponding neighbouring cells are replaced correctly. A method of first order accuracy is the *upwind* interpolation, given by

$$\phi_e = \begin{cases} \phi_P \text{ for } (\vec{u} \cdot \vec{n})_e > 0 \\ \phi_E \text{ for } (\vec{u} \cdot \vec{n})_e < 0 \end{cases} , \tag{3.16}$$

which is obviously dependent on the direction of the flux given by the velocity $\vec{u}$. This approximation of ϕ_e is equivalent to using a forward or backward difference in the finite difference method, depending of the direction of the velocity vector.

The most commonly used interpolation is the linear interpolation

$$\phi_e = \phi_E\lambda_e + \phi_P(1 - \lambda_e) \quad , \tag{3.17}$$

where λ_e is the linear interpolation factor defined by

$$\lambda_e = \frac{x_e - x_P}{x_E - x_P} \quad . \tag{3.18}$$

In the case of equidistant grids, the linear interpolation equation can be written as

$$\phi_e = \frac{\phi_E + \phi_P}{2} \quad . \tag{3.19}$$

Using a Taylor series expansion, it can be shown that the linear interpolation has a second order accuracy. Hence, using the mid-point rule (3.11) or (3.15) for the approximation of the integrals together with a linear interpolation (3.17) delivers an overall accuracy of second order.

Assuming a linear profile between the values of ϕ_P and ϕ_E, the gradient at the side 'e' can be approximated by central finite differences as

$$\left(\frac{\partial\phi}{\partial x}\right)_e = \frac{\phi_E - \phi_P}{x_E - x_P} \quad . \tag{3.20}$$

On an equidistant grid, this gradient approximation also has an accuracy of second order.

Higher order interpolations can also be constructed. One example would be the quadratic interpolation on an equidistant grid with an accuracy of 3rd order, where three neighbouring points are considered

$$\phi_e = \frac{6}{8}\phi_P + \frac{3}{8}\phi_E - \frac{1}{8}\phi_W \quad , \tag{3.21}$$

or a cubic interpolation on an equidistant grid with an accuracy of 4th order

$$\phi_e = \frac{27}{48}\phi_P + \frac{27}{48}\phi_E - \frac{3}{48}\phi_W - \frac{3}{48}\phi_{EE} \quad . \tag{3.22}$$

Using such higher interpolation techniques only makes sense if the corresponding integral approximations of higher order are applied, as the order of the approximation will otherwise not be raised.

The rest of the solution procedure is the same as in the case of the finite difference approach: the integral terms of the conservation equation have to be replaced by approximations for every control volume, the boundary conditions are formulated directly or as approximated equations, and an algebraic equation system can be set up and solved using an appropriate method.

Numerically speaking, it can be shown that methods of higher order tend to oscillate on coarse grids [FP02, Hir07], but will converge faster to a more accurate solution while refining the grids than do methods of lower order. Upwind approximations of first order are quite inaccurate and should not be used, whereas linear interpolation is a good compromise between accuracy, simplicity, and efficiency [FP02]. Thus, subsequently, this thesis uses linear interpolation in order to determine unknown values at different positions in grids, usually in combination with the mid-point rule.

3.3. Domain Decomposition and Non-Uniform Grid Treatment

If the domain becomes large, it will be inefficient (and due to memory limitations also impossible) to deal with the complete grid in one process. Thus, in order to prepare the system for a parallel computation, a domain decomposition approach becomes necessary.

The upper part of Figure 3.3 depicts a uniform grid which shall serve as the basis for performing a finite volume computation. In order to perform a domain decomposition, the large 4×2 grid is cut into two separate, non-overlapping, smaller 2×2 grids. The partial differential equations are now solved separately on both grids. Obviously, they cannot be solved independently, as this would mean solving two separate problems and, thus, the two grids have to be coupled together in order to compute the desired solution of the complete system.

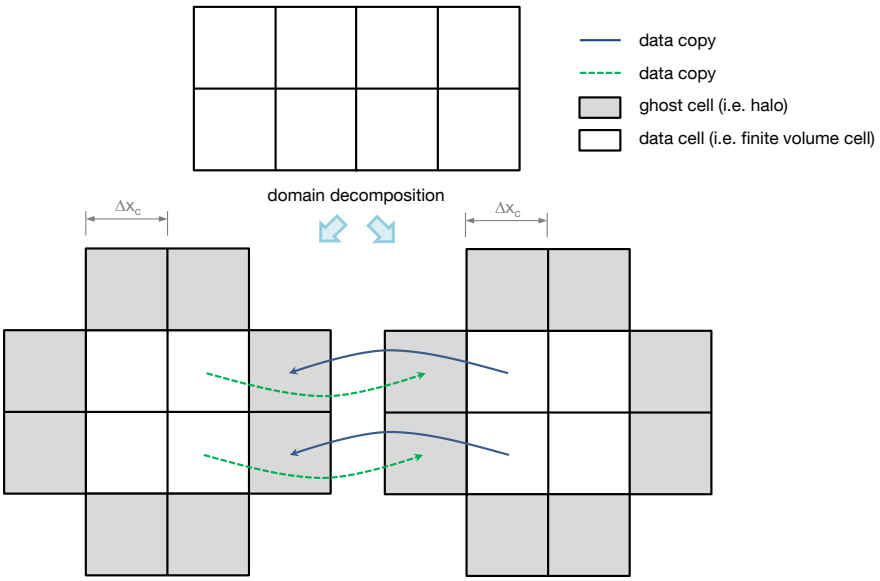


Figure 3.3.: Uniform grid (top) is decomposed into two separate grids using a ghost halo for data exchange (bottom).

Therefore, the two grids are padded with a layer of additional cells on each side, called *ghost cells* or *halo* and depicted in grey in the lower part of Figure 3.3. Although the two decomposed smaller grids do not overlap but share a common interface, the halos of the small grids do overlap. H. A. Schwarz showed in 1870 in his publication [Sch70] that a Laplace equation $\Delta u = 0$ can be solved on two overlapping grids in an iterative process by solving the equation on one grid and subsequently fixing the results obtained as boundary conditions on the second grid before solving the equation there. Schwarz proved the convergence of this method, whereas for other second order elliptic partial differential equations the convergence was proven by Mihlin [Mih51] in 1951. These methods of domain decompositions working on overlapping grids are henceforth known as *Schwarz methods*.

It is important to note that on one decomposed grid, only data cells (marked in white) are updated by the numerical scheme. The ghost cells (marked in grey) are used as 'artificial boundary conditions' for one computation step (also called 'owners-compute rule' cf. [DFF+03]). Thus, when applying a Schwarz method, the solution procedure can be described as iterations containing two separated phases: an update phase and a computation phase. Hence, the halos are updated with information from the neighbouring grids according to the arrows in Figure 3.3 before a computation step is executed, storing new values. This process is repeated until the solution converges.

Until now, only uniformly refined grids without any hanging nodes have been used. Thus, in the finite volume method, each cell had exactly one neighbouring cell per side. Figure 3.4 shows a non-uniform grid setup in the top part, where two hanging nodes occur. This domain is decomposed into two separate uniform grids and padded with a layer of ghost cells marked in grey in the lower part of the figure. Obviously, the data exchange is no longer as straightforward as in the uniform case as the overlapping halo portions have different sizes.

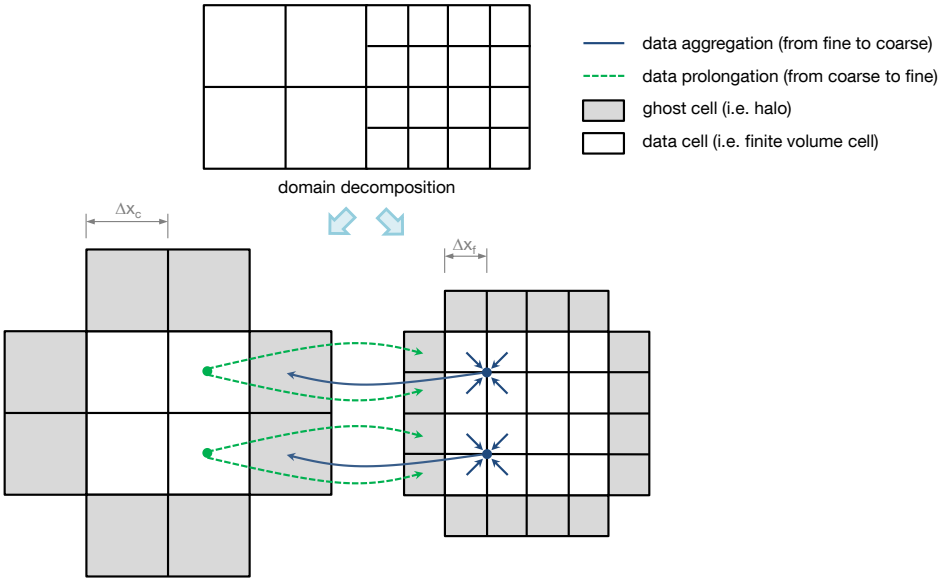


Figure 3.4.: Non-uniform grid (top) is decomposed into two separate uniform grids using a ghost halo for data exchange (bottom).

In order to achieve a consistent data exchange, a few operations have to be performed. To update data in the coarse grid halo originating from the finer one, data has to be aggregated to the same position where it would be stored in the coarse grid before it can be transferred to the corresponding opposite ghost cells. Supposing the data is stored in the cell centre, this means aggregating all data from cells which overlap with the corresponding coarse grid's ghost cell. When a bisection is chosen in two dimensions, this means four cells have to be aggregated to one value (in 3D it would be eight cells). It can be shown that, in a uniform regular grid, this aggregation degenerates to a simple averaging, as all the fine cell's surfaces have the same size.

In the opposite direction, a data prolongation has to be performed (denoted in Figure 3.4 as dashed arrows). If the data were only to be copied to the corresponding finer ghost cells without further treatment, incorrect results would be generated, as the data is supposedly stored at a different location in order to use the computational stencil. The prolongation uses the value and location of the coarse cell centre as well as the value and location of the fine cell centre in order to linearly interpolate the coarse grid value into the centre of the fine grid's ghost cell.

Once the halos are updated, the computation can be performed exactly as in the uniform case. The most important feature of this approach is the fact that the same stencil can be applied for both uniform grids. Hence, together with the modified update feature described above, a computation on a non-uniform grid can be performed using a uniform stencil, i. e. only one stencil describing the discretised partial differential equations has to be formulated and can be reused on all grids in the system.

3.4. Solution Procedures

Now that the mathematical problem is discretised either by finite differences or finite volumes, a solution can be computed. A very common method is the generation of a system of linear equations

$$\mathbf{A} \cdot \vec{f} = \vec{q} \tag{3.23}$$

where $\mathbf{A}$ is a matrix containing coefficients, $\vec{f}$ is the unknown quantity at every grid point, and $\vec{q}$ is the right-hand side of the equations containing given terms q_i independent of the quantity f_i.

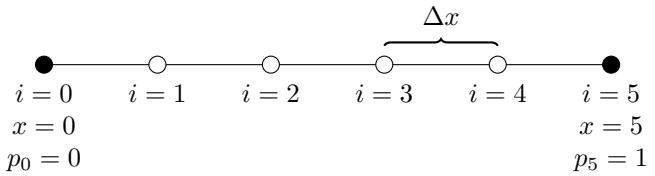


Figure 3.5.: Examplary 1D grid setup for demonstrating the assembly process.

In order to illustrate the assembly procedure, the solution of the 1D boundary value problem

$$\frac{\mathrm{d}^2 p(x)}{\mathrm{d}x^2} = q(x) \tag{3.24}$$

shall be considered on the domain $x \in [0, 5]$ with $p(0) = 0$ and $p(5) = 1$, and q a known function. Using the equidistant finite difference grid depicted in Figure 3.5 with central differences to approximate the second derivatives, the discretised problem for every grid point i can be written as

$$\frac{p_{i-1} - 2p_i + p_{i+1}}{\Delta x^2} = q_i \quad . \tag{3.25}$$

Together with the known (Dirichlet) boundary values, six equations can be written, which describe the complete system as

$$i = 0 \;:\; p_0 = 0 \quad , \tag{3.26}$$
$$i = 1 \;:\; p_0 - 2p_1 + p_2 = \Delta x^2 \cdot q_1 \quad , \tag{3.27}$$
$$i = 2 \;:\; p_1 - 2p_2 + p_3 = \Delta x^2 \cdot q_2 \quad , \tag{3.28}$$
$$i = 3 \;:\; p_2 - 2p_3 + p_4 = \Delta x^2 \cdot q_3 \quad , \tag{3.29}$$
$$i = 4 \;:\; p_3 - 2p_4 + p_5 = \Delta x^2 \cdot q_4 \quad , \tag{3.30}$$
$$i = 5 \;:\; p_5 = 1 \quad . \tag{3.31}$$

Substituting the known values (3.26) for p_0 and (3.31) for p_5 into Equations (3.27) and (3.30),

multiplying every equation with -1, and reformulating them in matrix form yields

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \cdot \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} = \begin{pmatrix} -\Delta x^2 \cdot q_1 \\ -\Delta x^2 \cdot q_2 \\ -\Delta x^2 \cdot q_3 \\ -\Delta x^2 \cdot q_4 + 1 \end{pmatrix} \quad , \tag{3.32}$$

which describes the complete discretised problem in the form of a system of linear equations in the form (3.23).

Neumann boundary conditions can be discretised by a forward or backward difference, which generates a new equation and can be integrated into the system of linear equations as well.

It should be noted here that special care has to be taken during the assembly process in order to achieve a well-conditioned matrix. In 1D the ordering is quite clear. In 2D and 3D a different ordering of the unknown vector $\vec{f}$ can have a drastic impact on the condition of the matrix and the subsequent solving process. As this problem is even more pressing in finite element-based discretisations, a lot of literature can be found describing strategies and remedies, such as [ZTZ05] for example.

This system of linear equations can be solved by direct solvers such as Gauss, or by iterative solvers such as Jacobi or CG (conjugate gradients). Depending on the matrix $\mathbf{A}$, not all methods can be used efficiently, or at all. For the Jacobi method, a regular, (strict) diagonally dominant matrix is required, whereas the CG requires a symmetric positive definite matrix. In the case of direct solvers, the system of equations is brought by elementary transformations into form such that the solution is apparent or easily deducible. Iterative solvers use repetitive procedures and require an initial solution (or initial state), thus converging towards an approximative solution. If the (partial) differential equation is non-linear, a linearisation has to be applied and the same methods can be used.

A second method for solving the system of linear equations can be deduced without the explicit assembly of a coefficient matrix. Here, a data structure such as the grid in Figure 3.1(b) can be used. An iterative method is updating the values $f_{i,j}$ in the grid in every iteration step, thus bringing them closer to the exact solution, which is also known as *matrix-free method*.

In order to solve the discretised Navier-Stokes equations, a multi-grid-like approach was used, belonging to the group of iterative solvers. This procedure is explained in detail in Section 4.6 after the data structure and further implementation details have been introduced. The basic ideas of the Jacobi solver and the standard multi-grid solver are presented in the following two sections, however.

Jacobi Solver

The probably simplest iterative solver is the Jacobi solver based on a matrix splitting scheme. Assuming a system of linear equations $\mathbf{A} \cdot \vec{f} = \vec{q}$ was assembled during a finite difference or finite volume discretisation, the matrix $\mathbf{A}$ can be decomposed into a lower, a diagonal, and an upper part

$$\mathbf{A} = \mathbf{L}_A + \mathbf{D}_A + \mathbf{U}_A \quad . \tag{3.33}$$

By substituting the matrix decomposition into the system of linear equations and rearranging the terms, the system can be written as

$$\mathbf{D}_A \cdot \vec{f} = (-\mathbf{L}_A - \mathbf{U}_A) \cdot \vec{f} + \vec{q} \quad . \tag{3.34}$$

Introducing the evaluation of the solution vector f at different iteration steps n and $n+1$, an iterative scheme can be constructed as follows

$$\vec{f}^{n+1} = \mathbf{D}_A^{-1} \cdot (-\mathbf{L}_A - \mathbf{U}_A) \cdot \vec{f}^{n} + \mathbf{D}_A^{-1} \cdot \vec{q} \quad . \tag{3.35}$$

The inverse of $\mathbf{D}_A$ can be computed easily, and it can be shown that for a diagonal dominant, regular matrix $\mathbf{A}$ resulting from a finite difference stencil, this scheme converges to the exact solution for any given start vector $\vec{f}^{0}$ (cf. [SK11], [Mei11]). Furthermore, a significant improvement in terms of convergence rate can be achieved by introducing a relaxation factor $\omega \in \mathbb{R}^+$. If $\omega < 1$ the method is called under-relaxed, and for $\omega > 1$ the method is called over-relaxed. The relaxed form of (3.35) is therefore

$$\vec{f}^{n+1} = \mathbf{D}_A^{-1} \cdot [(1-\omega)\mathbf{D}_A + \omega(-\mathbf{L}_A - \mathbf{U}_A)] \cdot \vec{f}^{n} + \omega \cdot \mathbf{D}_A^{-1} \cdot \vec{q} \quad . \tag{3.36}$$

This iterative procedure has to be repeated until the residual with respect to some norm $\|\cdot\|$ in iteration step n is

$$\|\vec{r}^{n}\| = \left\| \vec{q} - \mathbf{A} \cdot \vec{f}^{n} \right\| < tol \quad , \tag{3.37}$$

i. e. it is below a certain predefined tolerance. The choice of the norm is discussed later, as this is dependent on implementation details. One of these iteration steps is called a Jacobi step or a Jacobi smoothing step. If an under-relaxation approach is used, a step is usually called a damped Jacobi smoothing step.

The Jacobi solver usually requires many iteration steps until convergence is achieved with a reasonable accuracy. [BHM00] shows that high-frequency oscillations are smoothed quite fast, whereas low-frequency errors take a very long time to converge to a given accuracy. Therefore, in this work, the Jacobi solver is not used on its own, but rather as a building block for a multi-grid-like solver.

A similar method, called the Gauss-Seidel method, can be constructed by evaluating the

matrix decomposition in a different manner

$$(\mathbf{L}_A + \mathbf{D}_A) \cdot \vec{f} = -\mathbf{U}_A \cdot \vec{f} + \vec{q} \tag{3.38}$$

$$\vec{f}^{n+1} = \mathbf{D}_A^{-1} \cdot (-\mathbf{L}_A \cdot \vec{f}^{n+1} - \mathbf{U}_A \cdot \vec{f}^{n}) + \mathbf{D}_A^{-1} \cdot \vec{q} \quad . \tag{3.39}$$

This method can also be combined with an over-relaxation factor ω and is known as SOR (successive over-relaxation). Unfortunately, due to the mixture of iteration steps on the right-hand side, it is no longer easy to parallelise. Therefore, this method is not relevant for the rest of this thesis.

Geometric Multi-Grid Solver

The idea of geometric multi-grid solvers was introduced by Achi Brandt in the 70s as *multi-level method* in [Bra77]. Since then, it has been widely studied in literature, such as [BL11], [TOS01], [Hac10], or [BHM00]. The basic ideas of a geometric multi-grid solver will be briefly introduced here, before specific adaptations to the data structure which was actually used are introduced in Section 4.3 in order to apply a geometric multi-grid-like solver.

The multi-grid approach is based on two basic principles (cf. [TOS01]):

- *Smoothing*: many iterative methods (such as Jacobi or Gauss-Seidel) have a smoothing effect on the error if they are applied to discrete elliptic problems,
- *Coarse grid correction*: if a quantity is smooth on a fine grid, it can be approximated on a coarser grid, where it will behave in an oscillatory manner again.

The basic equation system $\mathbf{A} \cdot \vec{f} = \vec{q}$ can be rewritten in an equivalent form. By introducing the residual equation

$$\vec{r} = \vec{q} - \mathbf{A} \cdot \vec{f} \tag{3.40}$$

and the error equation

$$\vec{e} = \vec{f} - \vec{v} \quad , \tag{3.41}$$

where $\vec{f}$ denotes the exact solution and $\vec{v}$ is an approximation to it, an equivalent form of the equation system can be written as

$$\mathbf{A} \cdot \vec{e} = \vec{r} = \vec{q} - \mathbf{A} \cdot \vec{v} \quad . \tag{3.42}$$

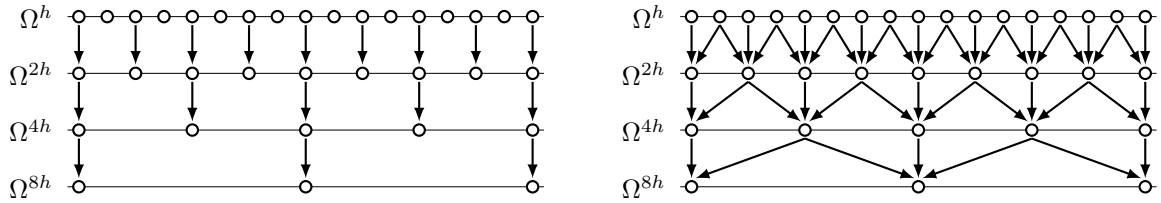
This equation suggests that a relaxation (i. e. smoothing) on the original system of equations $\mathbf{A} \cdot \vec{f} = \vec{q}$ with an arbitrary initial estimate for $\vec{f}$ is equivalent to smoothing on Equation (3.42) with initial estimate $\vec{e} = 0$.

[BHM00] shows that, by passing from a fine grid to a coarse grid, a smooth behaviour of a function on a fine grid is oscillatory on a coarser one. Thus, it can be relaxed again on the coarser grid until it is sufficiently smooth. This procedure is repeated until the grid is small enough to be solved by means of a direct solver, such as Gauss. The ‘passing’ in-between

the grids from fine (denoted by Ω^h) to coarse (denoted by Ω^{2h}) is called *restriction* and can be performed in different ways. Mathematically it can be written as an operator $\mathbf{I}_h^{2h}$ which is applied to a set of node points

$$\mathbf{I}_h^{2h} \cdot \vec{v}^h = \frac{1}{4} \begin{pmatrix} 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{pmatrix}_h = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}_{2h} = \vec{v}^{2h} \quad , \tag{3.43}$$

here as an example for a *full-weighting* restriction on a 1D grid with 7 points. The index h denotes the fine grid, whereas $2h$ denotes a coarser one. A typical full-weighting restriction can be seen in Figure 3.6(b). Another possibility of a restriction operator would be the pure *injection* depicted in Figure 3.6(a). In this case, no weighting from neighbouring points is used, making the injection the simplest form of a restriction operator. Thus, $\mathbf{I}_h^{2h}$ can be read as a restriction operator from a fine grid h to a coarser grid $2h$.



(a) injection (b) full weighting

Figure 3.6.: Restriction operations from fine grids to coarse grids in 1D.

Once the coarsest grid has been reached, the system can be solved by means of a direct solver as discussed previously. The solution of this coarse grid system is propagated to the finer grids. This process is called *prolongation* using the operator $\mathbf{I}_{2h}^h$

$$\mathbf{I}_{2h}^h \cdot \vec{v}^{2h} = \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}_{2h} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{pmatrix}_h = \vec{v}^h \quad . \tag{3.44}$$

If an interpolation is used as the prolongation operator, it fits very well to the full-weighting restriction as the prolongation operator is the transpose of the restriction operator multiplied by a constant factor c

$$\mathbf{I}_{2h}^h = c \cdot \left(\mathbf{I}_h^{2h}\right)^T \quad , \quad c \in \mathbb{R} \quad . \tag{3.45}$$

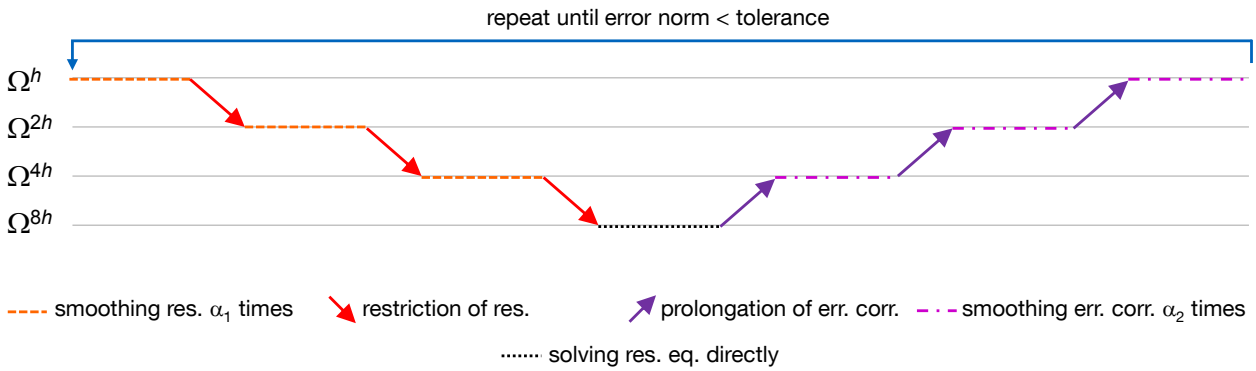


Figure 3.7.: Geometric multi-grid v-cycle solver concept.

Combining all the above-mentioned steps and ideas, a solver scheme can be established, which is depicted in Figure 3.7. The equation system is rewritten in equivalent form (3.42). A smoothing algorithm is then applied α_1 times and the residual is computed and restricted to a coarser grid. These steps are repeated until the coarsest grid has been reached, where the system will be solved directly resulting in an error correction term, which will be propagated to finer grids and applied to the already existing error correction term. After smoothing the equation again α_2 times, the prolongation step is repeated until the error correction has been applied on the finest grid. If the norm of the residual is below a predefined tolerance, the solution has been found, otherwise, all the steps are repeated again. Due to the shape of the method depicted in Figure 3.7, this scheme is referred to as *v-cycle*. The details of the v-cycle scheme are listed in Algorithm 3.1.

Algorithm 3.1 recursive v-cycle multi-grid

1: rewrite $\mathbf{A} \cdot \vec{v} = \vec{q}$ into equivalent form $\mathbf{A} \cdot \vec{e} = \vec{r}$
2: relax on $\mathbf{A}^h \cdot \vec{e}^h = \vec{r}^h$ α_1 times with the initial guess $\vec{e}^h = 0$
3: compute residual after relaxation: $\vec{r}^{h,*} = \vec{r}^h - \mathbf{A}^h \cdot \vec{e}^h$
4: restrict $\vec{r}^{2h} = \mathbf{I}_h^{2h} \vec{r}^{h,*}$ to a coarser grid
5: if $\Omega^{2h} \neq$ coarsest grid on level L goto 2, else goto 6
6: solve $\mathbf{A}^{Lh} \cdot \vec{e}^{Lh} = \vec{r}^{Lh}$
7: prolongate error correction $\vec{e}_c^h = \mathbf{I}_{2h}^h \vec{e}^{2h}$ to finer grid
8: correct error $\vec{e}^h \leftarrow \vec{e}^h + \vec{e}_c^h$
9: relax on $\mathbf{A}^h \cdot \vec{e}^h = \vec{r}^h$ α_2 times
10: if $\Omega^h \neq$ finest grid goto 7, else goto 11
11: apply error correction to approximation of solution $\vec{v} \leftarrow \vec{v} + \vec{e}^h$
12: if $\left\|\vec{r}^h\right\| = \|\vec{q} - \mathbf{A} \cdot \vec{v}\| > tol$ goto 1, else finished

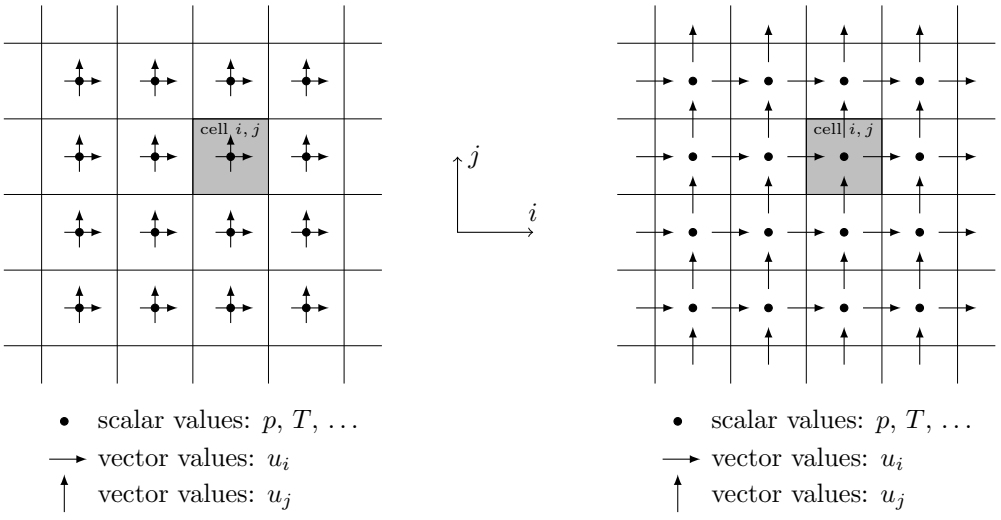
Depending on the restriction and prolongation operator, the pre- and post-smoothing factors α_1 and α_2 will have an influence on the convergence of the solution. In the recursive process, the restriction as well as the prolongation is repeated as many times as is necessary until the coarsest or finest grid respectively is reached. A general, mathematically sound convergence analysis of the method is presented in [TOS01] or [Hac10].

Alternative schemes to the v-cycle could be used, such as the μ-cycle, or the full multi-grid

v-cycle (FMG), but they will not be elaborated here. Interested readers might want to study [BHM00].

3.5. Collocated versus Staggered Grid Arrangement

There are several possibilities as to how the necessary primary variables such as velocity, pressure, temperature, or density can be distributed and stored on a discrete computation grid for CFD purposes. Typically, two setups are commonly used: collocated and fully staggered grid arrangements. Figure 3.8 depicts both setups.



(a) collocated grid arrangement (b) staggered grid arrangement

Figure 3.8.: Arrangement and storage of values in 2D grids.

Figure 3.8(a) shows the collocated arrangement, where all values are stored in the cell centre. In the fully staggered setup shown in Figure 3.8(b), scalar values are stored in the cell centre, whereas vector values such as velocities are stored on the corresponding sides, i. e. the velocity u_i in i-direction on the east and west side, the velocity u_j in j-direction on the north and south side, and the velocity u_k in k-direction on the top and bottom side (not depicted here for reasons of clarity). In the case of a staggered grid approach, it is mandatory that the cells are embedded in an additional layer of boundary cells, as otherwise one boundary value per velocity direction would be missing.

The advantage of the collocated arrangement is the uncomplicated treatment of the variables, as they are all located in one place. Visualisation as well as numerical algorithms tend to be easier to implement, as the values are closer to the formulations used in the mathematical description of Chapter 2. Furthermore, using collocated grid arrangements has huge advantages whilst using complex geometries or irregular domains [FP02]. The disadvantage is that, under certain circumstances, the pressure and the velocity field can become decoupled and non-physical pressure oscillations could arise from using a collocated arrangement.

This so-called odd-even-decoupling cannot happen in a staggered grid arrangement introduced by Harlow and Welch [HW65] due to a strong coupling of velocities and pressure. Nowadays, collocated grid arrangements offer more advantages than disadvantages if they are implemented correctly. Therefore, in the further course of this thesis, a collocated grid arrangement is used.

In order to avoid non-physical oscillations with the collocated grid arrangement, different control volumes are used, depending on the variable for which the equation is currently discretised.

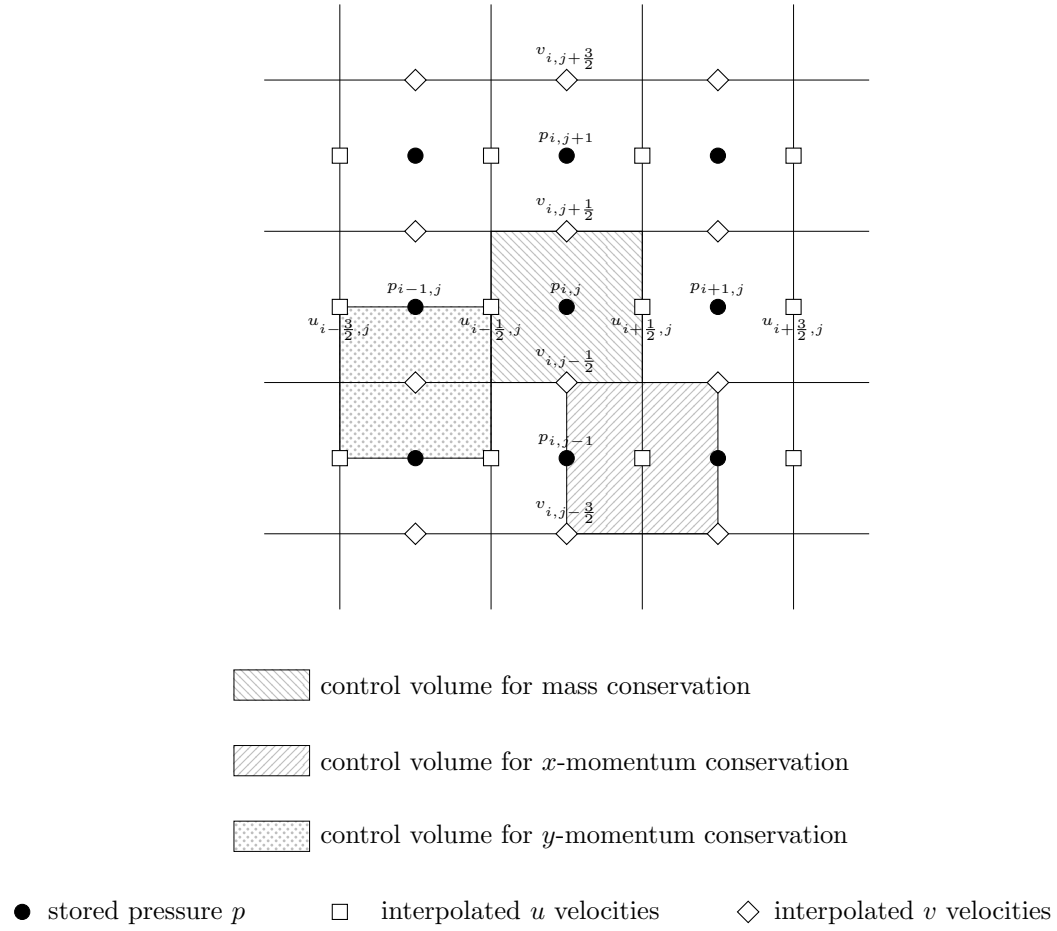


Figure 3.9.: Control volumes for discretisation using a collocated grid with virtual staggered values.

Figure 3.9 shows different 'virtual' control volumes. They are always centred around the respective variable. In the case of the discretisation of the momentum equation in y-direction, for example, the left-most highlighted control volume is used. Unknown values containing a fraction in the index are not stored directly but interpolated from values stored in cell centres. In this way, a strong coupling between velocity and pressure is again restored, and oscillations cannot occur.

3.6. Pressure Correction Method

As mentioned already briefly in Section 2.7, pressure correction methods were the first approach for solving the full, incompressible Navier-Stokes equations. For this work, the fractional step method, introduced by [Cho67] is applied. The temporal discretisation is realised using a forward Euler method, consisting of a first order forward difference approach in time. The spatial discretisation is introduced in a later stage and discussed after the Data Structure Section.

In order to introduce the basic ideas of the fractional step method, the momentum equation (2.20) in the direction i for an isothermal, incompressible Newtonian fluid flow without any external volume forces

$$\frac{\partial u_i}{\partial t} = -\nabla \cdot (u_i \vec{u}) + \nabla \cdot (\nu \nabla u_i) - \frac{1}{\rho} \nabla \cdot (p \vec{e}_i) \tag{3.46}$$

is recalled.

The method is based on an iteration between velocity and pressure. Applying the temporal forward Euler method to an intermediate time step and omitting the pressure term from (3.46) leads to

$$\frac{u_i^\star - u_i^n}{\Delta t} = -\nabla \cdot (u_i^n \vec{u}^n) + \nabla \cdot (\nu \nabla u_i^n) \quad , \tag{3.47}$$

where Δt denotes the time step size, the superscript ^{n} denotes the current time step n, and the superscript * an intermediate time step between n and $n+1$. The pressure term is now grouped in a second step

$$\frac{u_i^{n+1} - u_i^\star}{\Delta t} = -\frac{1}{\rho} \nabla \cdot \left(p^{n+1} \vec{e}_i\right) \quad . \tag{3.48}$$

Summing the two Equations (3.47) and (3.48), it is obvious that Equation (3.46) results, where the velocity is treated explicitly (i. e. at time step n) and the pressure implicitly (i. e. at time step $n+1$).

Using the divergence operator on the vector form of Equation (3.48) leads to

$$\frac{\nabla \cdot \vec{u}^{n+1} - \nabla \cdot \vec{u}^\star}{\Delta t} = -\frac{1}{\rho} \Delta p^{n+1} \quad . \tag{3.49}$$

As the pressure term needs to lead to a divergence-free velocity field at time step $n+1$ in respect of the continuity equation (2.9), the equation for determining the pressure at time step $n+1$ can be written as

$$\Delta p^{n+1} = \frac{\rho}{\Delta t} \nabla \cdot \vec{u}^\star \quad , \tag{3.50}$$

which represents a Poisson equation for the pressure.

To sum up, the steps necessary for the computation of one time step using a fractional step method include:

1. Compute the intermediate velocities $u_i^\star$ using Equation (3.47) for all directions i and every cell.
2. Compute the right-hand side of Equation (3.50) for every cell.
3. Solve the pressure Poisson equation (3.50).
4. Compute the velocity for time step $n+1$ using the resulting pressure in Equation (3.48).
5. Advance to the next time step.

This algorithm is the simplest available as the treatment of the time derivative is done explicitly. A slightly more complex variant can be implemented using a second order explicit multi-level Adams-Bashforth method [SK11]

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} = \frac{1}{2}\left[3f(t_n, \phi^n) - f(t_{n-1}, \phi^{n-1})\right] \quad . \tag{3.51}$$

Here, two levels are used, and values computed previously during time step n as well as time step $n-1$ can be reused in the computation of the next time step $n+1$. As the Adams-Bashforth method depends on two previous time steps, a different method, such as a standard forward Euler, has to be used for computing the values at the first time step, when no data for $n-1$ is available yet. The convective and diffusive terms are grouped together in the f function and dealt with explicitly. The pressure is still dealt with implicitly as in the previous method. This is currently the basis for the CFD code developed but could be further enhanced at some point by other methods.

A detailed analysis of other time derivative approaches can be found in [Fer98]. [KM85], for example, use a mixture of semi-implicit and explicit methods. The convective terms in Equation (3.46) are discretised using a second order explicit Adams-Bashforth method, whereas the viscous terms are discretised using a semi-implicit Crank-Nicolson method which eliminates some numerical stability problems. [CM94] use a similar approach but incorporate a predictor-corrector method.

3.7. Numerical Stability of the Explicit Euler Time Discretisation

As an explicit method for the time discretisation was chosen, it is imperative to discuss numerical stability criteria and determine under which conditions the method remains stable and can be applied. A very detailed stability analysis for the explicit finite difference method can be found in [PT83].

Here, a short and practical derivation is given by analysing the generic convection-diffusion equation (2.27) in differential form for a 1D case. By assuming a constant density property

ρ and a constant generic diffusion property Γ in time and space, the generic convection-diffusion equation without any internal volume sources q_ϕ can be written as

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot (\phi \vec{u}) + \frac{\Gamma}{\rho} \nabla \cdot (\nabla \phi) \quad . \tag{3.52}$$

By applying the chain rule to the convective part, introducing the continuity equation, and introducing an explicit forward Euler discretisation for the time derivative with a time step Δt, Equation (3.52) can be rewritten in non-conservative form as

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} = -\vec{u} \cdot \nabla \phi + \frac{\Gamma}{\rho} \nabla \cdot (\nabla \phi) \quad . \tag{3.53}$$

Using central differences and an equidistant finite difference grid with grid spacing Δx in 1D, the discretised form becomes

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = -u \frac{\phi_{i+1}^n - \phi_{i-1}^n}{2\Delta x} + \frac{\Gamma}{\rho} \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{(\Delta x)^2} \quad . \tag{3.54}$$

This equation can be rewritten by reordering the terms to give

$$\phi_i^{n+1} = (1 - 2d)\phi_i^n + \left(d - \frac{c}{2}\right) \phi_{i+1}^n + \left(d + \frac{c}{2}\right) \phi_{i-1}^n \quad , \tag{3.55}$$

and by introducing two dimensionless parameters

$$d = \frac{\Gamma \Delta t}{\rho (\Delta x)^2} \quad \text{and} \quad c = \frac{u \Delta t}{\Delta x} \quad . \tag{3.56}$$

The parameter d represents the ratio between the time step Δt and the characteristic diffusion time $\rho (\Delta x)^2 / \Gamma$ which is necessary to transport a quantity via diffusion over an interval Δx. The parameter c represents the ratio between the time step Δt and the convection time $u/\Delta x$ which is necessary to transport a quantity via convection (with a positive velocity u) over an interval Δx. The parameter c is called the *Courant number* and is a very important quantity in fluid dynamics using explicit time stepping methods.

By assembling the algebraic equations for every grid point into a matrix $\mathbf{A}$ and rewriting the problem, a *von Neumann* stability analysis can be performed on the problem which involves an analysis of the eigenvalues of matrix $\mathbf{A}$. The complete analysis is performed in [FP02]. It can be shown that, if no diffusion is present (i. e. $d = 0$), the method is always unstable, and cannot be applied at all. If no convection is present (i. e. $c = 0$) the method is partially stable and restrictions on d have to be imposed in order to use it in a safe manner. According to the analysis, all coefficients of Equation (3.55) have to be positive in order to ensure a stable method (cf. [FP02]). Thus, the coefficient of ϕ_i^n imposes a restriction on the time step by forcing $d < 0.5$, thus

$$\Delta t < \frac{\rho (\Delta x)^2}{2\Gamma} \quad . \tag{3.57}$$

The second coefficient in front of ϕ_{i+1}^n does not restrict the time step itself, but by forcing $c < 2d$, this results in

$$Pe := \frac{\rho u \Delta x}{\Gamma} < 2 \quad . \tag{3.58}$$

Thus, the Péclet number Pe, which represents the ratio of convective transport to diffusive transport in a medium, has to be restricted.

Courant and Friedrichs analysed this problem in the 1920s and proposed using an upwind-difference for the convection term and central differences for the diffusive term. Thus, Equation (3.53) can be discretised as

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = -u \frac{\phi_i^n - \phi_{i-1}^n}{\Delta x} + \frac{\Gamma}{\rho} \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{(\Delta x)^2} \quad , \tag{3.59}$$

which can be written with the parameters c and d from Equation (3.56) as

$$\phi_i^{n+1} = (1 - 2d - c)\phi_i^n + d\phi_{i+1}^n + (d + c)\phi_{i-1}^n \quad . \tag{3.60}$$

The coefficients of ϕ_{i+1}^n and ϕ_{i-1}^n are always positive and cannot produce non-physical oscillations. In order to have a positive coefficient for ϕ_i^n, the following condition has to be fulfilled

$$\Delta t < \left(\frac{2\Gamma}{\rho (\Delta x)^2} + \frac{u}{\Delta x} \right)^{-1} \quad . \tag{3.61}$$

If the convection can be neglected, the stability condition is the same as for Equation (3.57). In the case of negligible diffusivity, however, the stability criterion is

$$c < 1 \quad \text{or} \quad \Delta t < \frac{\Delta x}{u} \quad , \tag{3.62}$$

stating that the method is only stable if the Courant number is less than 1. Further details can be found in [CFL28].

The method applied in Equation (3.55) is first order in time and second order in space. Equation (3.57) dictates that, if the grid resolution is reduced by a factor of two, the time step has to be reduced by a factor of four.

As the full 3D Navier-Stokes equations differ slightly from the pure 1D generic convection-diffusion equation dealt with above, the following restrictions can be imposed for our code (see also [GDN98]):

- restrictions due to CFL conditions (i. e. convection)

$$\Delta t_{c1} < \frac{\Delta x_1}{|u_{1,max}|} \quad , \quad \Delta t_{c2} < \frac{\Delta x_2}{|u_{2,max}|} \quad , \quad \Delta t_{c3} < \frac{\Delta x_3}{|u_{3,max}|} \quad , \tag{3.63}$$

- restrictions due to diffusion ([TM94])

$$\Delta t_{d\nu} < \frac{1}{2\nu} \left(\frac{1}{(\Delta x_1)^2} + \frac{1}{(\Delta x_2)^2} + \frac{1}{(\Delta x_3)^2} \right)^{-1} \quad , \tag{3.64}$$

- restrictions due to thermal diffusion

$$\Delta t_{d\alpha} < \frac{1}{2\alpha} \left(\frac{1}{(\Delta x_1)^2} + \frac{1}{(\Delta x_2)^2} + \frac{1}{(\Delta x_3)^2} \right)^{-1} \quad . \tag{3.65}$$

In order to determine the maximal possible time step, the minimal value of all above conditions is used

$$\Delta t = \tau \cdot \min \left(t_{c1}, t_{c2}, t_{c2}, t_{d\nu}, t_{d\alpha} \right) \quad , \tag{3.66}$$

where $\tau \in]0, 1]$ is a safety factor.

As the stability criteria above are necessary but not sufficient, a safety factor τ is introduced in order to reduce the possible time step size. The velocity-dependent criteria (3.63) have to be checked after every time step, whereas the velocity-independent criteria (3.64) and (3.65) need only to be checked once (assuming constant ν and α).

3.8. Data Structure

The design of an efficient and lean data structure is a very crucial part of the complete simulation pipeline, especially when parallelisation strategies are involved. In the design process, a very close relation exists between the numerical treatment discussed in this chapter and implementation aspects discussed in Chapter 4. The basic ideas are presented here, whereas precise implementation details regarding the data structure are discussed later on.

The data structure of the code was designed with a parallel implementation in mind. Thus, the efficiency is not tuned for a serial execution on one machine, but rather for runs on multiple machines using a distributed memory approach. As the data structure itself forms the basis of the code, but also influences the numerical treatment drastically, it was designed in a very early stage of the project. As changes and adjustments were necessary over time in order to take additional, as well as previously unplanned features, into account, the data structure design evolved over time. This can be seen in the scientific publications over the years, describing the data structure at several implementation stages. The following publications all mention parts of the data structure to some extent: [FMR11a], [FMR11b], [FMR12], [FMR13a], [FMR13b], and [FMR14]. Although the description was correct for its time, the following section gives an overview of the current and latest stage of development.

The data structure is based on non-overlapping[1], block-structured, hierarchical, orthogonal,

[1] the actual grids are non-overlapping, while the padded ghost cells around those grids overlap as described in Section 3.3

regular grids. It is based on a logical, purely topological grid management (called 'l-grids'), and data grids containing the actual data (called 'd-grids').

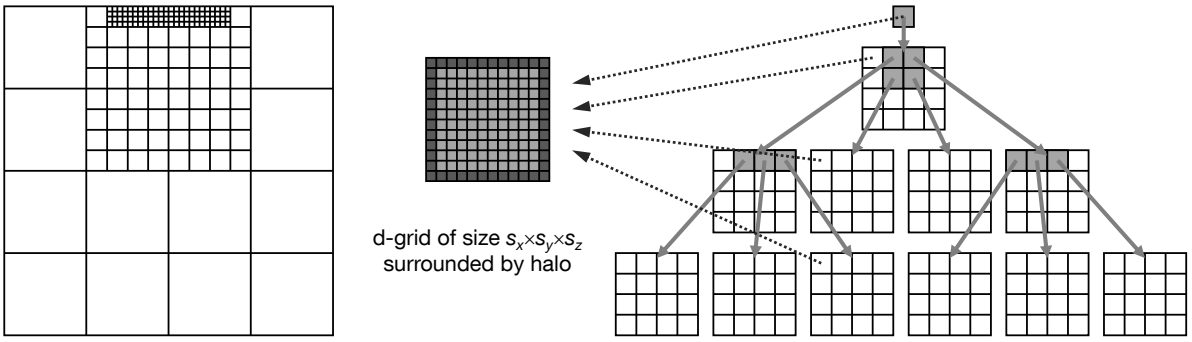


Figure 3.10.: Hierarchical Data Structure: logical l-grid structure without d-grids (i. e. data grids) (right-hand side), construction of the logical structure (left-hand side) with d-grids linked to every logical l-grid (middle) [FMR13b]

Figure 3.10 gives an overview of both data structure parts. On the right-hand side, the nested construction of the non-overlapping grids can be seen. A root l-grid – located per definition at depth 0 – is refined by r_x^t, r_y^t, r_z^t in the respective directions. The new resulting child l-grids can be refined again by r_x^s, r_y^s, r_z^s until the desired depth d is reached. Currently, r_i^t and r_i^s can be chosen independently out of the interval $[1, 7]$. Furthermore, while creating the newly refined child l-grids, the creating l-grid will be defined as their parent l-grid. Hence, each l-grid can only have one parent l-grid.

If the refinement in a direction is marked to be '1', it will not be refined at all. Hence, all 2D computations are performed as *pseudo* 2D computations in a 3D domain with a refinement $r_z^{t,s} = 1$. The choice of a different refinement level for the root l-grid and subsequent l-grids is very practical in cases where a non-cubic domain, such as a channel, is to be computed, as the domain structure can be better represented by such a non-uniform sub-division. On the lower levels, this is not allowed in order to keep the structure as simple as possible. On the left-hand side of Figure 3.10, the logical grid structure is overlaid and the refined regions of higher depth can be seen quite well for a refinement of (4,4,1) for top-level and sub-levels. There exist a total of 177 l-grids in the setup depicted here.

The second major part of the data structure are the d-grids (i. e. the data grids). Every d-grid stores the necessary variables such as velocities, pressure, or temperature values in a matrix of cells. Thus, a cell is comparable to one finite volume or control volume. It is surrounded by a halo of ghost cells necessary for the proper functioning of the structure. One of these d-grids can be seen in the middle of Figure 3.10. It has a size of s_x, s_y, s_z (not counting the halo) which can be chosen arbitrarily with one constraint: in order to avoid non-conforming setups s_i has to be divisible without remainder by r_i^s and by r_i^t. This restriction guarantees that coarse d-grid boundaries will always coincide with refined d-grid boundaries. Each l-grid contains exactly one link to one d-grid.

For the above-depicted setup, there exist 177 d-grids linked to their corresponding l-grid. In the case of a uniform refinement, the number of data cells existing in the domain can be

easily computed by

$$N_{\text{cells}}^{\text{uniform}} = \prod_{i \in \{x,y,z\}} r_i^t \cdot (r_i^s)^{d-1} \cdot s_i \quad , \tag{3.67}$$

where d is the maximal depth of the refinement.

By applying this type of data structure together with a finite volume approach for the spatial discretisation, a few special points can be noted here.

One d-grid is constructed in such a way that it has an equidistant orthogonal spacing. In this case it can be shown that the finite volume approach using the mid-point rule and linear interpolation degenerates into a finite difference approach. Thus, on one d-grid, a simple finite difference scheme such as a six-point stencil in 3D can be used as outlined in detail in Section 3.3. This allows a separation into two phases: a computation phase and a communication phase. In the computation phase, the finite difference approach in the form of a stencil, for example, is evaluated on every d-grid. In the communication phase, a halo update is performed filling the ghost cells with neighbouring values in order to apply a Schwarz method. Usually, a communication phase is only necessary for a computation running on a distributed system, but for this structure it is required even in serial cases. Details about the communication patterns, the distribution strategy, and other implementation aspects are given in Chapter 4. For a complete list of definitions and terms used the reader is also referred to Appendix A.

3.9. Numeric Discretisation of the Navier-Stokes Equations

As the data structure is now properly introduced, the numerical treatment of the spatial discretisation can be addressed. The temporal discretisation of the projection method was introduced in Section 3.6. Furthermore, a virtual staggered approach is used where the necessary values on the boundaries of cells have to be determined via interpolation. A very good and detailed approach with 2D equations is presented in Chapter 12 in [Hir07].

As seen in Equation (3.50), a Poisson equation for the pressure $\Delta p = f$ has to be solved. According to [GDN98] or [FP02], the efficient solution of this equation is mandatory, as most time will be spent on solving this equation, given that an explicit pressure correction method is applied.

As in the current implementation for the pressure Poisson equation, the computations are performed matrix-free immediately on the d-grid, i. e. the finite difference equations are applied immediately on the variables of a data cell rather than assembled into matrix form. In this case, for an equidistant 3D discretised d-grid with central differences, a ω-damped

Jacobi smoothing step can be written as

$$p_{i,j,k}^{n+1} := (1-\omega)p_{i,j,k}^{n} + \omega \cdot \left(\frac{p_{i+1,j,k}^{n} + p_{i-1,j,k}^{n}}{(\Delta x)^2} + \frac{p_{i,j+1,k}^{n} + p_{i,j-1,k}^{n}}{(\Delta y)^2} + \frac{p_{i,j,k+1}^{n} + p_{i,j,k-1}^{n}}{(\Delta z)^2} - f_{i,j,k} \right) , \quad (3.68)$$

where f represents the right-hand side of Equation (3.50).

In the case of the standard Jacobi solver introduced in Section 3.4, Equation (3.68) can be executed in a loop until the norm of the residual is below a given threshold. In the case of a multi-grid-like approach, as highlighted in Algorithm 3.1, Equation (3.68) can be used as a smoother. The integration of the data structure into the multi-grid-like approach is discussed in Section 4.6.

3.10. Pseudo-Algorithm for the CFD Simulation

Combining all the steps from above, as well as the information gathered in the previous sections, a pseudo-algorithm describing a complete CFD computation which applies a fractional step method and uses an explicit time stepping scheme can be summarised in Algorithm 3.2.

Algorithm 3.2 CFD simulation using a fractional step with an explicit time stepping scheme

1: set $t \leftarrow 0$
2: assign initial values to $\vec{u}, p, T, \rho, \ldots$
3: **while** $t < t_{end}$ **do**
4: select time step Δt according to stability conditions (Section 3.7)
5: compute intermediate velocities $\vec{u}^{\star}$ using Equation (3.47) enhanced by the Boussinesq approximation (2.34) in the case of a thermal simulation
6: compute the right-hand side of the pressure Poisson equation (3.50) using $\vec{u}^{\star}$
7: solve the pressure Poisson equation (3.50) using a multi-grid-like approach
8: update the velocities to the next time step using Equation (3.49)
9: compute the temperature problem (2.31) explicitly in the case of a thermal simulation
10: write results for visualisation purposes every n^{th} simulation step
11: set $t \leftarrow t + \Delta t$
12: **end while**

4. Towards a Massive Parallel Implementation of the Navier-Stokes Equations

As the mathematical treatment as well as the basic numerical discretisation schemes were discussed in the previous chapters, implementation aspects can be tackled now. They comprise much more than a pure choice of programming language. This section will deal with unique l-grid identification across different processes, parallel data distribution strategies, communication and load balancing schemes, and the treatment of the boundary condition, amongst other things.

4.1. Unique Grid Identification Mechanisms

As mentioned in Section 3.8, the data structure is built up using l-grids and d-grids. In order to use the previously mentioned algorithms, it is important to be able to address a specific l-grid on a specific process[1]. Hence, a unique l-grid identification is necessary, and the decision was taken to encode the l-grid identification into a long integer consisting of 64 bits. The first 32 bits (ordered from right to left) describe an l-grid tag, identifying the l-grid on a process. The second 32 bits identify the rank of the process where the l-grid resides. In a MPI-based implementation[2], the rank and tag can be used to uniquely identify any l-grid on any process immediately.

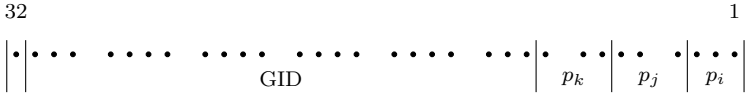


Figure 4.1.: Bit encoding scheme for the tag part of the unique l-grid identification (UID).

The tag of an l-grid consists of a 32-bit integer into which certain information is encoded bitwise, as depicted in Figure 4.1. Three bits are used for the identification of each location in the parent l-grid in i, j, or k-direction. Thus, an l-grid knows immediately its location in its respective parent l-grid. 22 bits are used as l-grid ID or 'GID'. The GID has to be unique on one process and, thus, it is possible to store up to $2^{22} - 1 = 4{,}194{,}303$ l-grids per

[1] Once an l-grid is uniquely identified, its corresponding d-grid is also known, as they are directly linked.
[2] Message Passing Interface (MPI) Standard: `http://www.mpi-forum.org`

process. The last bit is reserved, as most current MPI implementations use this bit internally for their own purpose. The root l-grid on the top-most level will always have l-grid ID 0 and the position in the parent l-grid (p_i, p_j, p_k) will be ignored. On the other hand, using three bits per direction is a trade-off, as now each direction can be refined up to a maximum of $2^3 - 1 = 7$. But as numerical issues suggest, using a recursive subdivision of more than five is physically not really feasible any more and, thus, is no limitation within the proposed approach.

Using the above convention for numbering l-grids over different processes is quite convenient while using MPI libraries, as `rank` and `tag` can be used immediately in all communication calls.

4.2. Geometry Description and Generation

In order to set up complex geometry scenarios, a valid geometry description needs to be introduced for the data structure presented in Section 3.8. Furthermore, this process needs to be fast in case a change occurs and the geometry discretisation in the domain has to be updated.

The general idea of a space-tree generator as described in [Sam90] or [Mun05], for example, is to generate a volume-based model from a surface-based (triangular) geometry description. Usually in 3D, an octree is generated using a 'voxel' as primitive, where each dimension is bisected, resulting in $2^3 = 8$ child voxels (hence the name octree) which can again be subdivided recursively. The decision as to whether a subdivision should be performed is based on geometric intersection tests using the triangular geometry description. As soon as one triangular surface is inside, touches, or crosses one voxel it is recursively subdivided, until no triangle is located inside any more, or until a given maximal depth is reached.

To this end, [Mun05] describes a fast octree generator based on multiple geometric tests with varying costs and a queue-based treatment. The idea is that, while applying inexpensive geometric tests, a lot of possible candidates can be discarded before more expensive intersection tests need to be applied. The basic algorithm of the octree generator is modified and adapted to an ld-grid-based hierarchical data structure and presented in Algorithm 4.1. An ld-grid is defined as the abbreviation for one l-grid and its corresponding linked d-grid (see also Appendix A).

The ld-grid generation algorithm is deeply integrated and internally linked to the above-described data structure, which is created 'on the fly' while the algorithm proceeds and treats the input geometry. It is basically structured into two stages. During the first stage, the l-grid layout is created. Once the maximal depth has been reached, the d-grids are created and linked to their l-grids. In a second step, the flags for 'boundary' have to be set in the leaf d-grids on the lowest level. Therefore, the algorithm is executed again after a subdivision of any 'boundary' voxels into the number of data cells present in the d-grids. After the second run, the boundary cells are clearly marked and can be used in a subsequent

Algorithm 4.1 queue-based hierarchic ld-grid generator

```
1: for operationMode in {createLogicalGrids, createDataGrids} do
2:     if operationMode = createLogicalGrids then
3:         read in geometric triangular model and store triangles tri in a list tri_l
4:         setup empty in_q and out_q queues
5:         create root voxel and associate all indexes to all faces in tri_l to voxel->triIdx_l
6:         push root voxel to in_q
7:     else if operationMode = createDataGrids then
8:         in_q ← ∅
9:         for all voxel in out_q do
10:            if voxel->isGeometricBoundary = true then
11:                remove voxel from out_q and subdivide voxel into s_x · s_y · s_z children
12:                copy remaining voxel->triIdx_l to all children
13:                push all child voxel to in_q
14:            end if
15:        end for
16:    end if
17:    while in_q ≠ ∅ do
18:        get next voxel from in_q
19:        voxel->colour ← undefined
20:        for all tri indexed from voxel->triIdx_l do
21:            if voxel bounding box ∩ tri bounding box = ∅ then
22:                remove tri index from voxel->triIdx_l
23:            end if
24:        end for
25:        if voxel->triIdx_l ≠ ∅ then
26:            for all remaining tri indexed from voxel->triIdx_l do
27:                if one of tri vertexes lies inside voxel then
28:                    voxel->colour ← grey
29:                else if one of the tri edges intersects the voxel then
30:                    voxel->colour ← grey
31:                else if the plane containing the tri vertexes intersects the voxel then
32:                    voxel->colour ← grey
33:                else
34:                    remove tri index from voxel->triIdx_l
35:                end if
36:            end for
37:        end if
38:        if voxel->colour ≠ grey then
39:            voxel->isGeometricBoundary ← false
40:            push voxel to out_q
41:        else
42:            if maxDepth reached or operationMode = createDataGrids then
43:                voxel->isGeometricBoundary ← true
44:                push voxel to out_q
45:            else if operationMode = createLogicalGrids then
46:                subdivide voxel into r_x · r_y · r_z children
47:                copy remaining voxel->triIdx_l to all children
48:                push all child voxel to in_q
49:            end if
50:        end if
51:    end while
52: end for
53: return out_q
```

fluid flow computation. In order to safely determine for a given geometry if a cavity is reachable, i. e. whether the non-boundary cells are inside or outside of the given geometry, a flood filling algorithm such as described in [FvDFH96] can be used. A seed point in the fluid region is selected by the user and the algorithm checks which area is connected to this seed point. As an additional feature, this algorithm can detect, if the inflow and outflow boundaries for a channel are connected by one common fluid region, for example. If this is not the case, the chosen resolution is too coarse and a higher depth has to be used.

A very interesting point of Algorithm 4.1 is that it can be parallelised quite easily as the main `while` loop has no dependencies and, thus, can be executed nicely in parallel. A performance analysis of the original, unmodified octree algorithm was presented in [FM10] and showed very promising run times of approx. 30 seconds for voxelising 12.7 million triangular input elements using an octree depth of 10 in a non-parallel computation producing over a billion voxels on a standard desktop computer.

4.3. Communication Schemes for Parallel Distributed Data Structures

The concept of the data structure, as introduced in Section 3.8 and depicted in Figure 3.10 is very well-suited to parallelisation by distributing different ld-grids over multiple processes. Questions such as load balancing and communication schemes are discussed in the next sections before the multi-grid-like solver integrated directly into the parallel data structure is addressed.

Assuming the different ld-grids are distributed amongst the computing processes, data updates have to be performed at regular intervals in order to exchange data from neighbouring d-grids into the ghost halos of the d-grids for the subsequent computation phase using a finite difference stencil (cf. Schwarz method in Section 3.3).

In the case of a non-parallel computation, all data resides on the same process and, thus, synchronisation is implicitly given by the ordering of the lines of code. It is imperative that data has to be available in the ghost halo before the stencil can use it for computing update values, as otherwise undefined states may result, which must be avoided.

In the case of a parallel computation, the same prerequisites have to be fulfilled: data must have been exchanged completely in all ghost halos of one d-grid, before a computation can start. Thus, synchronisation mechanisms have to be implemented, preventing a computation before all necessary data has been exchanged. In a distributed parallel computation the synchronisation is implemented by explicit message exchanges between different processes.

Using the aforementioned data structure, the communication process can be split into three different communication stages, which have to be synchronised internally, in order to prevent undefined states. This is most important for adaptive d-grids, as the data structure is responsible for the flux conservation over d-grid boundaries.

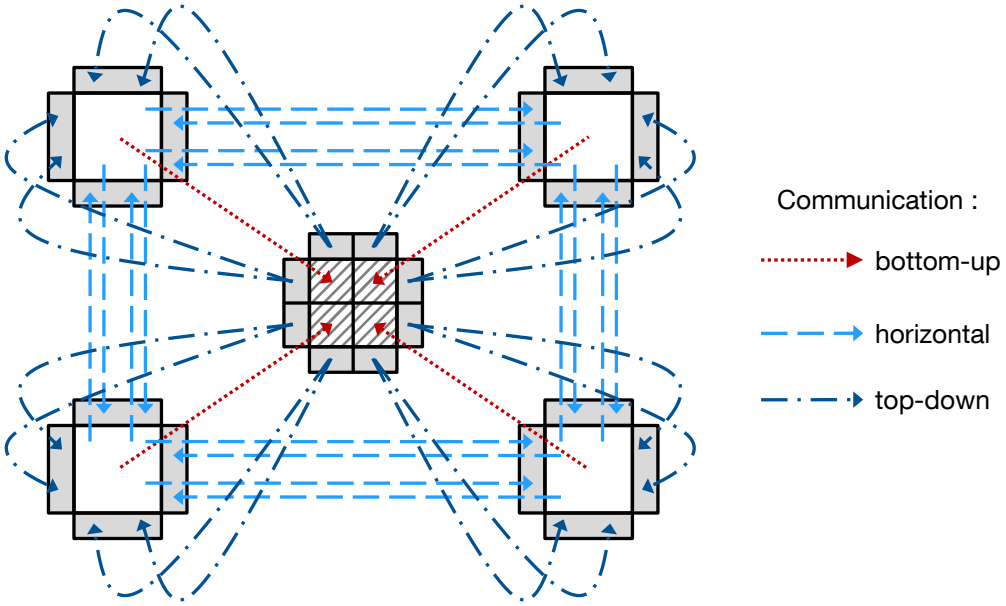


Figure 4.2.: Communication structure for complete data exchange over one level.

The three different communication stages can be described by a bottom-up, a horizontal, and a top-down communication phase. All phases are depicted in Figure 4.2 using different colours and line styles. The middle hatched d-grid resides on level d having four refined d-grid children at level $d+1$. The grey areas surrounding the d-grids themselves represent the ghost halos, and the d-grids have an internal size of s_x, s_y, s_z.

The communication process involves the following three phases.

- *Bottom-up communication*: In the bottom-up communication phase, data is averaged and sent upwards from children to parents where it is stored in the parents' corresponding data cell. In the case of a 3D bisection with $r_x = r_y = r_z = 2$, for example, eight child values are averaged into one parent d-grid value. This communication phase is depicted with red dotted arrows in Figure 4.2.

  Technically, this synchronisation process is achieved by using a mixture of blocking and non-blocking MPI communication calls: every l-grid issues a non-blocking `MPI_Irecv` call for every child (if it exists) followed by a blocking `MPI_Waitall` command. As soon as all data from all children has been received, it averages its own data and sends it upwards using a blocking `MPI_Send` command. Every process traverses its list of l-grids in a bottom-up manner allowing averaged data to be propagated upwards until the root l-grid is reached. At first, only l-grids having no children are not blocked, compute averaged values, and send them upwards. This procedure will process the l-grid hierarchy in a bottom-up manner and ensure a correct synchronisation.

  Theoretically, in the case of a pure, uniform l-grid refinement up to a certain level d_{max}, this step would not be necessary and could be ignored, as no other l-grid in a later stage will be dependent on data from further refined neighbouring l-grids. Nevertheless, this step is also done in the case of a uniformly refined l-grid setting, as a lot of

additional functionality can be gained, such as a *sliding window* visualisation, which will be explained in detail in Section 5.2.

In an adaptive case, however, data from a further refined neighbouring l-grid is necessary in the third (top-down) communication phase and it is necessary to ensure that this data is updated and available during that stage of communication.

- *Horizontal communication*: In the horizontal communication phase all data can be exchanged directly by neighbouring l-grids at the same depth. During this phase no special consideration has to be given to synchronisation as this exchange is independent of the order of exchange. It could even be interleaved with the third communication phase described in the next paragraph. Horizontal communication is marked by light blue dashed arrows in Figure 4.2. As can be seen from the diagram, horizontal communication always copies information from data cells to ghost cells. No data is copied into data cells.

  Technically, simple non-blocking `MPI_Irecv` commands are issued together with blocking `MPI_Send` commands and a subsequent `MPI_Waitall` to be sure that all data has arrived. Data will then be copied from internal receiving buffers to the actual data fields.

- *Top-down communication*: The third and last communication phase deals with a top-down communication, where all ghost cells are updated which have not been touched during the horizontal communication phase. This applies to all boundary ghost cells which do not have direct neighbours, as they would be outside of the domain, and also to l-grids which have no neighbour at the same level, as in the case of an adaptively refined l-grid. This phase updates only ghost cells and does not touch data cells at all. In order to reduce the communication cost to a minimum, data values from a coarser d-grid are sent to a finer d-grid, stored in a temporary buffer, and replicated on the deeper d-grid for updating all ghost cells. Here, synchronisation is again necessary in order to prevent data inconsistencies. The hierarchic tree has to be traversed from top to bottom, hence the name: top-down communication.

  Technically, the synchronisation is ensured by a mixture of blocking and non-blocking communication calls. All child l-grids issue non-blocking `MPI_Irecv` calls for each side where no horizontal communication has occurred yet, and post a subsequent `MPI_Waitall` call. Starting at the root (which does not receive data from above), data is sent to the children with a `MPI_Send` call, until the deepest l-grid is reached.

This concludes the complete exchange process and a subsequent computation phase can start. Technically, communication can be restricted to necessary quantities such as only velocity values, or only pressure values. This is very useful, as in the different internal steps of the computation phase not all values have to be updated at once (or at all for this matter).

When taking the different communication stages into account, the synchronisation procedure might seem to have a large overhead and synchronisation cost. Measurements presented later on show, however, that the benefits outweigh the cost. Before concentrating on performance

measurement results, however, a few other points such as ld-grid distribution and load balancing have to be discussed.

4.4. Data Storage Strategies and Neighbourhood Servers

During every phase of the data exchange, parent l-grids and child l-grids, as well as neighbouring l-grids, have to be known in order to be able to send and receive messages to/from the right destination. In the case of a statically adaptive l-grid arrangement, where all allocations of l-grids to processes are known a priori, these relations can be stored on every l-grid statically and do not have to be updated during run time. Obviously, a load balancing has still to be done in order to achieve a good performance. Load balancing strategies are discussed in Section 4.5.

As one design goal of the data structure was a dynamical ld-grid distribution during run time in case some areas need to be resolved using a higher resolution – either because geometry was moved in a computational steering approach, or due to fluid flow requirements, such as high local velocity gradients or refinements of eddies, for example – a different approach has to be applied. In general, two major strategies are frequently used for dynamical load balancing: using a local or a global view. The decision as to which strategy is best suited is problem dependent, and both have their advantages and drawbacks.

If only a local view of the system is known, a diffusion process, such as that proposed by [Cyb89], has to be applied. Load imbalances are always distributed to direct neighbours. This process has to be repeated iteratively until an optimal solution has been found, as in every step a load imbalance can only be propagated to direct neighbours. If more information about the global system is (or would be) available, the diffusion constants can be adapted and a faster convergence towards the optimal solution can be reached. The strong aspect of this method is the non-necessity of complete overall global knowledge of the system. On the other hand, moving ld-grids from one process to another becomes more complicated in a highly adaptive setup, as all connecting l-grid neighbours, parents, and children have to be informed of the UID change. In the worst case, collective communication has to be used for this purpose, which limits performance.

The second possibility is a global view strategy. In this case, a global server keeps track of all l-grid information by constructing an internal hierarchical tree structure, for example. If a load imbalance occurs, the global process can find an optimal solution without the need for several iteration steps. The drawback of this solution is that the global process can (and will) become a bottleneck regarding performance, as every process has to communicate with the server at some point.

By considering the possible advantages, and keeping the drawbacks in mind, the decision to use a global strategy was made. Furthermore, bottleneck avoidance was taken into consider-

ation. Figure 4.3 shows the general concept of the so-called *neighbourhood servers* embedded into the simulation framework.

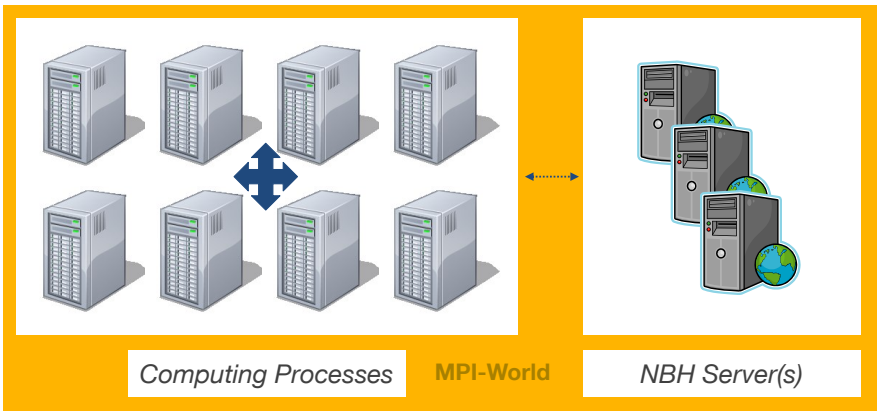


Figure 4.3.: Neighbourhood server (NBH) concept. Thickness of arrows indicates the amount of data transferred over this connection.

4.4.1. Concept of the Neighbourhood Server

The general idea is to deploy additional processes, called neighbourhood servers, that manage the bookkeeping of the l-grid allocations to processes. Technically, MPI uses so-called *communicators* to group processes together in order to provide collective communication capabilities. Further details can be found in the MPI technical documentation[3]. In a standard MPI program, the standard communicator `MPI_COMM_WORLD` (grouping all running processes) suffices for communication. In this case, however, the communicators are split into two disjoint groups. The first group contains the computing processes, where the global ranks range from $[0; m-1]$, m being the number of computing processes. The second group contains the neighbourhood servers with ranks ranging from $[m; n-1]$, n being the total number of processes available. All necessary collective communication between the computing processes is now using its own communicator, while interprocess communication between computing processes and neighbourhood servers is still done using the global ranks of the `MPI_COMM_WORLD` communicator.

In the simplest case, only one neighbourhood server will be available. During the domain construction process, the l-grids created on the computing processes are continuously registered in the NBH server structure which tracks their specific location as well as any topological connections to the parent l-grid and possible child l-grids. As soon as all l-grids are created on all processes, a load balancing is performed by the neighbourhood server which will redistribute ld-grids onto processes in order to divide the computational load as well as communication aspects in an optimal way.

After rebalancing the ld-grids, the neighbourhood relation cache has to be refreshed on all processes. This is done once in the beginning and is therefore not regarded as performance

[3] `http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf`

critical. Every process queries for each side of each l-grid which UID its neighbour has and caches it in an array. Thus, each process knows with which process it has to exchange data during the communication phase. In the case of a static, i. e. non-run-time adaptive computation, the neighbourhood server can be used to generate UIDs and perform the initial load balancing. After this point, no further data exchange with the NBH server will be necessary.

In a dynamic, i. e. run-time adaptive simulation, the neighbourhood server is of much greater importance. If a refinement of an ld-grid was ordered by an adaptive fluid criterion or by a user changing some geometry parameters, the neighbourhood server has to reflect these changes and implement them over all processes, wherever necessary. To this end, the neighbourhood server has a changing queue, which stores coded orders in an unsigned character together with two UIDs (old UID before migration and new UID after migration). Every process has to contact the neighbourhood server at a regular interval and check the internal queue for changes involving l-grids stored at its position. Then the process decodes the order and refines, coarsens, or transmits ld-grids to another process. After the implementation of the change, the neighbourhood server removes this element from the treatment queue. If a huge imbalance occurs between the processes, the NBH server performs a rebalancing and issues ld-grid migration orders to the treatment queue.

A very crucial point was mentioned above and should be highlighted here: the maximal data flow between the neighbourhood server and the processes is one unsigned character for a coded order plus a maximum of two UIDs, i. e. on 64-bit architectures a maximum of 17 bytes. If an ld-grid migration from one process to another is to occur, the data is sent immediately from one process to another without passing over the neighbourhood server. The amount of transferred data is also indicated in Figure 4.3 by the size of the arrows.

Grid Traversal for Neighbour Identification

One of the major tasks of the neighbourhood server is to answer queries of processes to identify UIDs of neighbouring l-grids. The querying process sends the UID of the l-grid and the direction in which the neighbour should be retrieved to the neighbourhood server. Once the neighbourhood server receives the request, it identifies the required l-grid using the given UID. The tag part of the UID is decoded, and the position (p_i, p_j, p_k) in the parent l-grid is extracted (cf. Figure 4.1) and increased or decreased in the respective searching direction. If the increased or decreased position is still valid (i. e. a sibling of the l-grid), the neighbouring l-grid is found and its UID returned to the querying process. Otherwise, the procedure has to be repeated for the parent l-grid and the given search direction until a common parent l-grid can be found (in the worst case the root l-grid). After a sibling in the common parent l-grid has been able to be identified, the path has to be retraced until the starting level is reached. If this l-grid is present, the UID is sent to the querying process, otherwise an empty result is sent, as there is no refined l-grid on the same level. This tree traversal procedure is repeated for all l-grids where neighbours are to be identified. If, at some point, the search

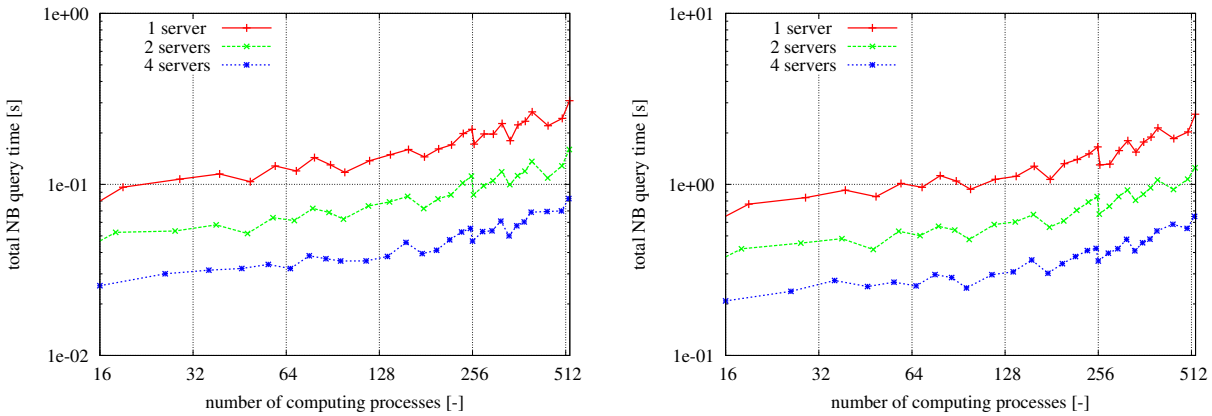
direction were to cross the main domain boundary, an 'out-of-domain' result would be sent back, as there are no neighbours on the outside of the domain.

This functionality is also used in the case when the l-grid structure is to be restructured in such a way that neighbouring l-grids have a difference in depth of maximal one level (cf. tree balancing). Depending on the numerical treatment of the partial differential equations, this is of the utmost importance, as certain numerical schemes only work stably if the depth differences are not too large. If a refinement of $r_x = r_y = r_t = 4$ is to be chosen in 3D, a level difference of two would mean that $(4^2)^2 = 256$ refined cells would all have one neighbouring cell, which might introduce large errors. In this case one of the outcomes of such a neighbourhood search is the level difference to the next higher neighbouring l-grid, if no l-grid on the same level could be found. Using this information, a list of l-grids which have to be refined can be generated, and the tree can be restructured, i. e. balanced. Obviously, this has to happen iteratively as some changes in the l-grid structure might have ripple effects on other parts. But when using the search strategy and the refinement list, it can be shown that the maximum number of times it has to be repeated is of the order of the depth of the structure.

4.4.2. Bottleneck Avoidance Using Multiple Neighbourhood Servers

Even if most of the data transfer does not involve the server directly, it can easily become a bottleneck in a dynamical setting, as every process has to communicate at regular intervals with the server in order to obtain updates, or if a massive number of processes is to be used (over 32,768 processes, for example). One solution proposed and described in [FMR12] is to use multiple neighbourhood servers. As most of the queries are 'read-only', they can be executed in parallel without any necessary synchronisation. Every order changing the neighbourhood server state, however, has to be synchronised which again introduces an overhead. This synchronisation problem is quite similar to the cache-coherency problem in computer hardware architecture. There, one possible solution is the so-called *MESI protocol* [CSG99]. MESI is an abbreviation for states called *exclusive **M**odified*, ***E**xclusive unmodified*, ***S**hared*, and ***I**nvalid*. Different transitions change the states of cache lines in a local processor cache in combination with a write-back cache policy. The same analogy can be used with multiple neighbourhood servers. Assume we have four servers all storing the same data. As long as nobody is modifying the data, the servers can use their local data for answering 'read-only' queries. This can all happen in parallel without any synchronisation. The same holds for the MESI protocol. As long as all cache lines are in the shared state, they can be used directly from cache without reloading them from main memory. If a particular processor is now locally modifying a cache line, which was in the shared state, all other copies of that cache line at different processors become invalid and the state of the modified cache line is changed to exclusive modified. The other processors are not allowed to use this specific cache line any more. The same will happen with the servers. As soon as one server receives a command changing the data, it sends an interrupt message to all other servers.

They will stop treating external requests until the changes from the initiating server have been implemented. After the changes have been realised, all servers resume their normal query answering tasks, until the next change is requested.



(a) depth 4: 4,681 l-grids (b) depth 5: 37,449 l-grids

Figure 4.4.: Complete query time of neighbourhood information for different numbers of computing processes and different numbers of servers. (Computed on 'snb' and 'snb3' partitions at TUM MAC-Cluster (cf. Appendix B.2.2)). Settings: uniform refinement using bisection until given depth; d-grid size of 12^3.

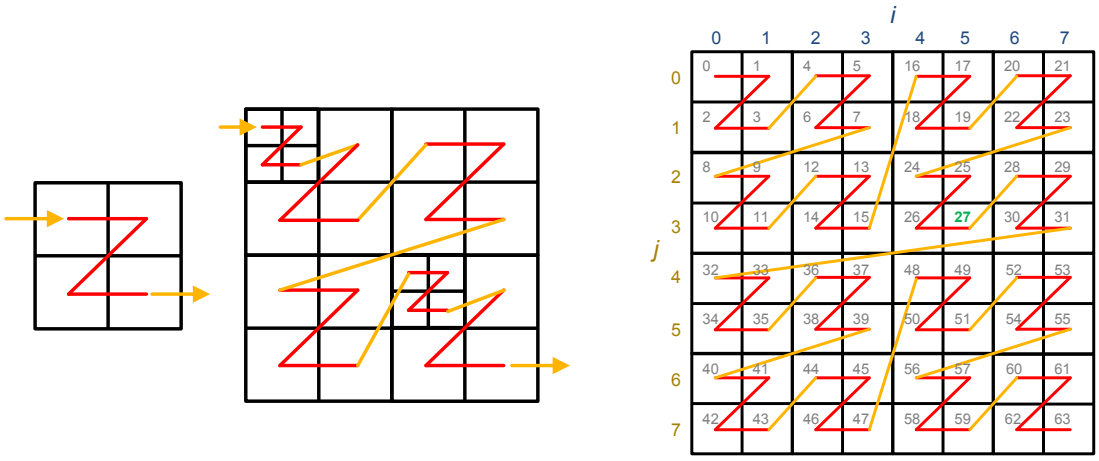
In order to get an idea about performance and time consumption of the neighbouring querying functions, some performance tests were done using the TUM MAC-Cluster. Figure 4.4 shows the total query time necessary to refresh all neighbouring information on all l-grids using different numbers of computing processes. The ideal behaviour would be a constant querying time, as the number of queries remains constant. Instead we observe a slight increase in the querying time resulting from the fact that each process has to query for less data, but more processes are trying to get information from the neighbourhood server at the same time. As the data access for the queries is read-only, multiple servers can answer the queries concurrently. To this end, parts of the computational processes have different responsible neighbourhood servers. By using two or four servers, for example, the querying time can be decreased by a factor of two or four, respectively, regardless of the number of l-grids. The fluctuations in the querying time can be explained by a non-ideal distribution of ld-grids to processes over the computing domain. As the same characteristics can be seen in all results, these fluctuations are independent of the neighbourhood server querying processes. For 4,681 l-grids the average query time for refreshing all neighbourhood caches is around 0.1 seconds (cf. Figure 4.4(a)); for 37,449 l-grids it takes approximately 1 second using only one NBH server on the MAC-Cluster (cf. Figure 4.4(b)).

4.5. Distribution Strategies and Load Balancing

The previous section described that the ld-grid migration is organised by the neighbourhood server issuing migration commands. The question still remains as to which strategy to use

for deciding about placement and ordering. It is quite obvious from the description of the communication strategies that a load imbalance will have a severe impact on performance.

This problem is a well-known and also a well-analysed one in computer science and mathematics and can be reformulated as how to map a multi-dimensional space (in this case 3 dimensions) onto a linear array while keeping locality information preserved to some extent. So-called *space-filling curves* are one solution to tackle this problem.



(a) geometric construction of a Lebesgue curve on an adaptive l-grid (geometric primitive on the right-hand side, adaptive example on the left-hand side)

(b) uniform l-grid distribution with enumeration according to Morton's index.

Figure 4.5.: Load balancing strategy by applying Lebesgue's curve and Morton's indexing for a 2D example.

There are multiple space-filling curves, such as Hilbert, Lebesgue, Peano, or Sierpiński curves. Each of them has its advantages and disadvantages on certain grids. In [Bad13] a very detailed review of the mathematical properties and application ranges is described. For the current work, the Lebesgue curve [Leb04] (also called Z-order curve, depicted in Figure 4.5(a)) was selected due to the possibility of a fast index computation by the bit-interleaving introduced by Morton in [Mor66].

In order to compute a linearisation of the curve according to the *Morton index*, a prerequisite has to be met: the indices of the dimensions $i, j, k \in \mathbb{N}^+$ (in 3D) must have the same length l_m in binary notation:

$$l_m = \lceil \max(\log_2(i_{max}), \log_2(j_{max}), \log_2(k_{max})) \rceil \quad . \tag{4.1}$$

Hence, the indices in binary notation have to be padded with zeros until the given length l_m is achieved. Afterwards, an interleaving of the binary notation can be performed, thus generating the Morton index.

In order to compute the Morton index m for an l-grid depicted in Figure 4.5(b) in 2D for

$i, j < 2^3$, the generic bit interleaving can be written as

$$i = (ib_2\ ib_1\ ib_0)_{\text{binary}}\ ,\ j = (jb_2\ jb_1\ jb_0)_{\text{binary}} \quad \rightarrow \quad m = (jb_2\ ib_2\ jb_1\ ib_1\ jb_0\ ib_0)_{\text{binary}} \tag{4.2}$$

where i, j, m are in binary form and ib_x and jb_x represent the x-th bit. As an example consider

$$i = 5_{\text{dec}} = (101)_{\text{bin}}\ ,\ j = 3_{\text{dec}} = (011)_{\text{bin}} \quad \rightarrow \quad m = (011011)_{\text{bin}} = 27_{\text{dec}} \quad , \tag{4.3}$$

which corresponds to the result in Figure 4.5(b) constructed according to the Lebesgue curve. The same technique applies to any number of dimensions. Hence, the Morton index in the implemented 3D code can be computed very fast by simple bit-shifts and binary operations.

The load balancing strategy mostly consists of applying the above-described methodology: while linearising the logical l-grid structure according to the Lebesgue curve using the Morton index, the pointers to l-grids are appended to a list starting with the root l-grid. The final list is divided into (nearly) equal parts assigned to each process. The neighbourhood server responsible for this process then orders an ld-grid migration for all ld-grids which do not yet reside on their dedicated processes. Thus, an initial as well as an intermediate load balancing step during run time can be easily performed.

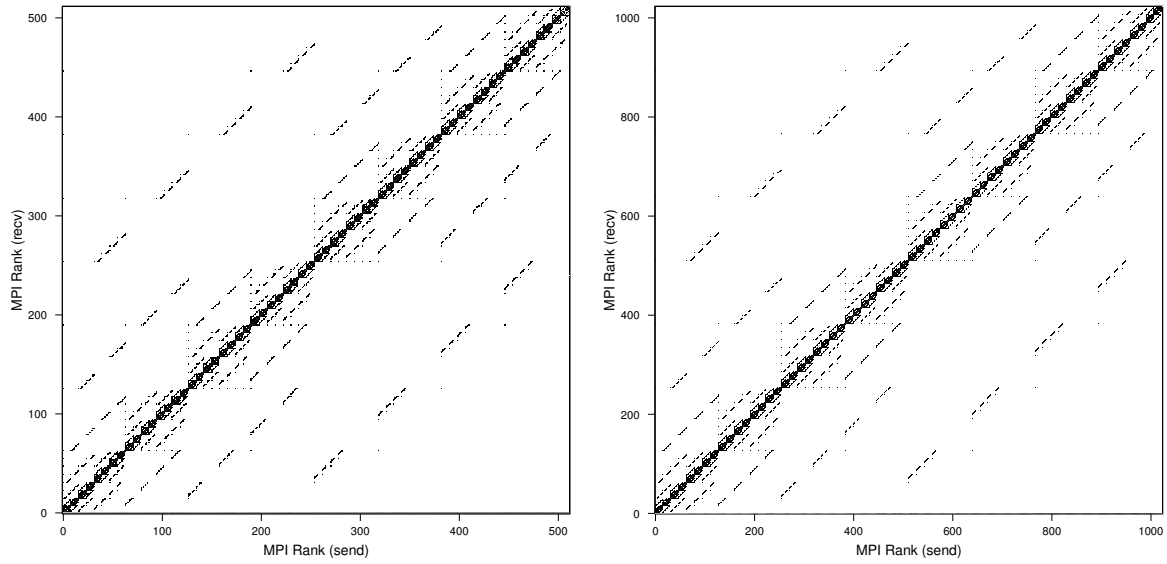
Performance Measurements of Communication Procedures

In order to determine whether the space-filling curve applied is adequate for the distribution of the ld-grids over different processes, measurements were performed on the 'Shaheen' Blue Gene/P system in the Kingdom of Saudi Arabia (cf. Appendix B.2.5) and on the Blue Gene/P system at the Universitatea de Vest din Timişoara (UVT) in Romania (cf. Appendix B.2.4) using the integrated performance monitoring tool IPM[4], which is able to monitor communication between processes.

Figures 4.6 and 4.8 show such communication diagrams. Here, the x-axis depicts the sending rank, and the y-axis the receiving rank. If rank 1 sends to rank 10, a mark will be placed where $x = 1$ and $y = 10$ meet, for example. Moreover, the communication time is encoded in the grey scale of the marks. The darker a mark, the more communication occurs between the corresponding processes.

In Figures 4.6(a) and 4.6(b) the same number of unknowns (a total of $(2^6 \cdot 4)^3$) was distributed over 512 and 1,024 processes, respectively. The neighbourhood server resides at the last process (rank 511 and 1,023, resp.). Here, every process communicates with the NBH server at some point, which answers the requests as described in Section 4.4. Hence, these lines are both completely filled. As no dynamic rescheduling was applied here, bottleneck avoidance was not an issue, and only one NBH server was used.

[4]available at `http://ipm-hpc.sourceforge.net` (supported by the NSF under award number 0721397)



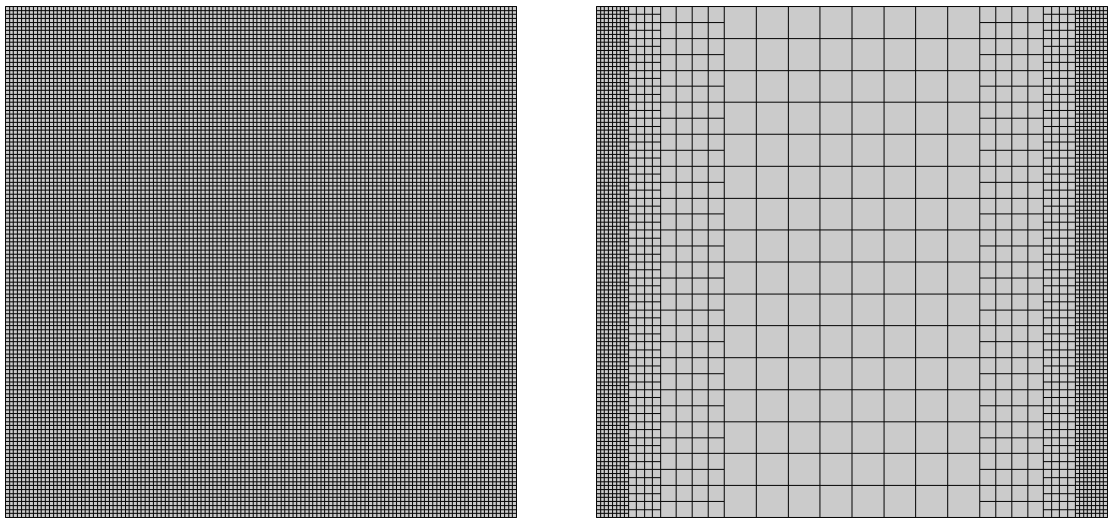
(a) 512 processes (b) 1,024 processes

Figure 4.6.: Communication pattern measurements on Shaheen for a 3D domain of fully refined l-grids with a refinement level (2,2,2) up to depth 6.

Furthermore, the communication is mostly concentrated along the diagonal axis, meaning that only ranks close to each other, i. e. physically neighbouring l-grids, communicate (with some 'off-diagonal' exceptions, which are mostly formed due to links to parents residing on different processes). Yet, one has to bear in mind that close ranks do not necessarily mean short communication pathways. Depending on the mapping of the real network topology to the virtual MPI topology, results can be different. The measured systems showed good results for the Lebesgue space-filling curve applied, however. Furthermore, distributing the same number of unknowns and l-grids to twice as many processes produces similar communication patterns, indicating that the space-filling curve works as intended and is sufficient for our purpose.

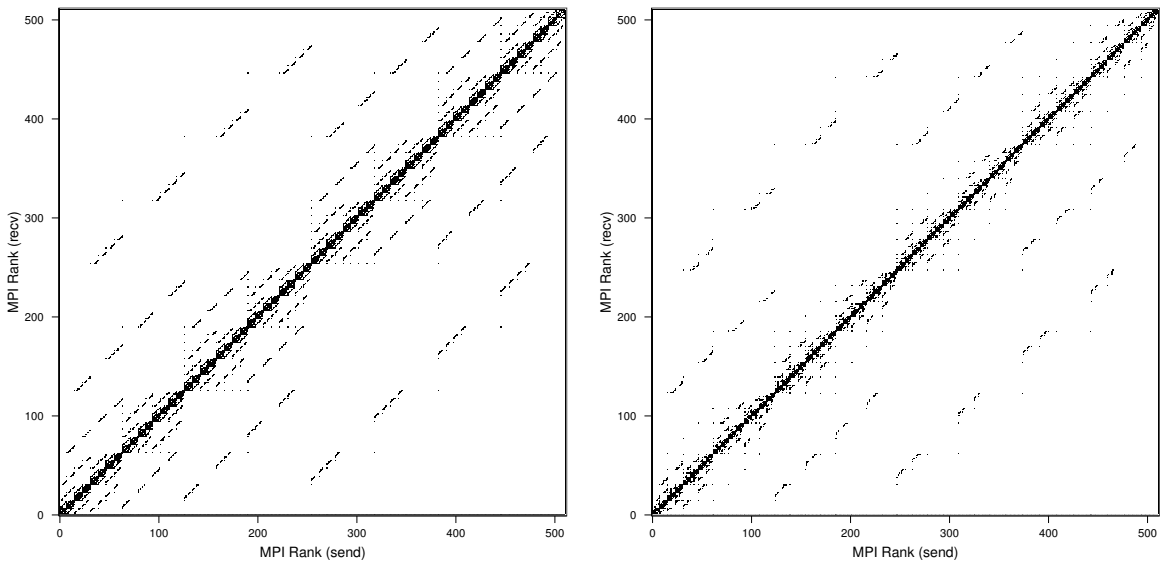
Contrary to the examples in Figure 4.6, Figure 4.8 shows the comparison of a uniform l-grid refined to level 5 (Figure 4.8(a)) with an adaptively refined l-grid (Figure 4.8(b)) always distributed over 512 processes. The adaptive l-grid is refined uniformly to level 2 and then gradually refined 3 times towards the boundaries, resolving the boundary layer with the same resolution as the uniform l-grid (cf. Figure 4.7). This means that the adaptive setup has considerably fewer l-grids than the uniformly refined setting. Nevertheless, the communication patterns stay diagonally dominant as in the uniform case. Hence, the distribution process works also with adaptively refined geometry settings as expected.

As the distribution and communication patterns have now been analysed and described in detail, one can come back to the previous question of the efficiency of the three-layered communication structure (bottom-up, horizontal, and top-down phase). Using space-filling curves for the distribution of the l-grids to different numbers of processes, the communication times for a total exchange step of all ghost-cell values over all levels can be measured. A



(a) uniform setup at depth 5 ($u5$) (b) adaptive setup at depth 5 ($a2+3$)

Figure 4.7.: Representation of the discretisation of a 2D cut through the 3D domain with a refinement level (2,2,2): left-hand side shows a uniform refinement to depth 5 while right-hand side shows a uniform refinement to depth 2 with 3 adaptive refinement steps towards the boundaries.



(a) uniform setup at depth 5 ($u5$) (b) adaptive setup at depth 5 ($a2+3$)

Figure 4.8.: Communication pattern measurements on Shaheen for 512 processes for a 3D domain with a refinement level (2,2,2) refined once uniformly to depth 5 and once uniformly to depth 2 with 3 adaptive refinement steps towards the boundaries (cf. Figure 4.7).

uniform l-grid distribution is the worst case in terms of communication time according to our communication phases, as this means the maximum amount of data to transfer.

Figure 4.9 depicts the wallclock time in seconds for a total exchange using different depths and numbers of l-grids as well as unknowns. The wallclock time is measured for three different systems: the TUM MAC Cluster (cf. Appendix B.2.2), SuperMUC (cf. Appendix B.2.3), and Shaheen (cf. Appendix B.2.5). All systems used a bisection in all three dimensions in the l-grid hierarchy and a d-grid size of $16 \times 16 \times 16$. Thus, depth 5 consists of 37,449 l-grids and 153.4 million cells, accumulating to a total of 1.38 billion variables which have to be exchanged. Depth 6 has 299,593 l-grids and 1.23 billion cells, which add up to a total of 11.04 billion variables to exchange. Depth 7 has 2.3 million l-grids, 9.8 billion cells and, thus, 88 billion variables to exchange. Depth 8 – the deepest one measured – consists of 19 million l-grids, 78.5 billion cells and 707 billion variables to transfer. All examples at the respective depths on all the systems used exactly the same settings.

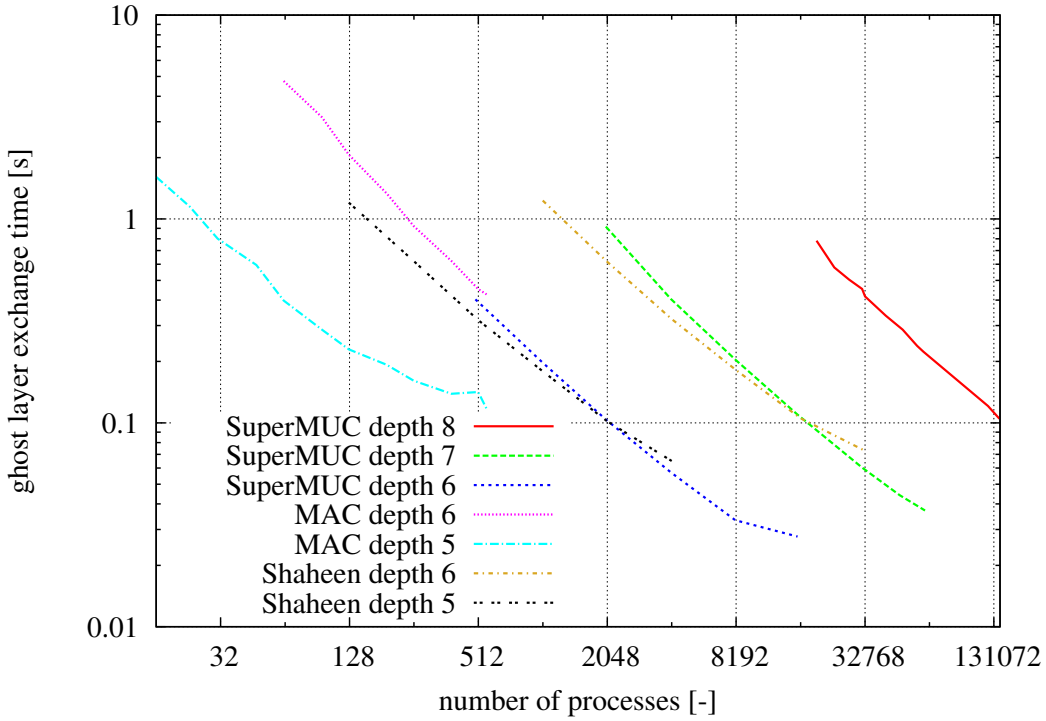
In Figure 4.9(a), a decrease in communication time can be observed with an increasing number of computing processes. As expected, the MAC cluster and the SuperMUC measurements deliver similar results as the underlying hardware is exactly the same, and the network connection in the MAC cluster is similar to SuperMUC's intra-island wiring.

For all systems it can also be seen that depth $d+1$ needs longer to communicate than depth d, which is obvious as around eight times more data has to be transferred. Furthermore, it can be observed that the slope of the different depths is similar. Hence, the communication scales perfectly with the number of l-grids per process. In order to better analyse the behaviour for different numbers of l-grids per process, Figure 4.9(b) plots the ghost layer exchange time over the numbers of l-grids per process. As the space-filling curve distributes the number of l-grids equally over the different computing processes, the average number of l-grids per process can simply be computed by dividing the total number of l-grids by the number of computing processes.

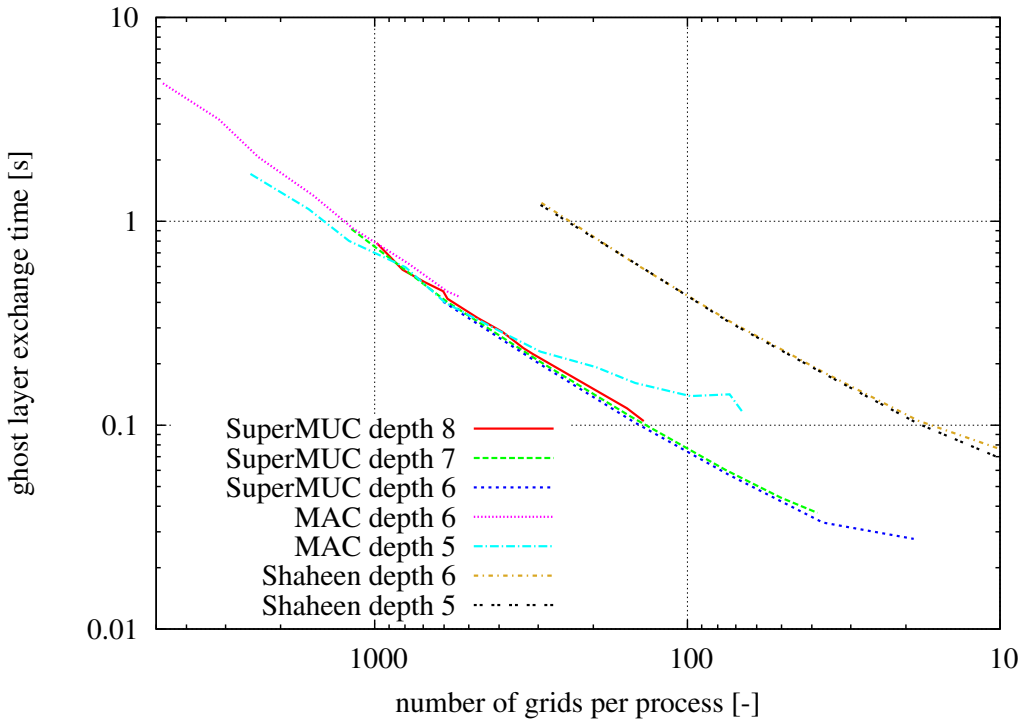
For all systems it can be observed that the distribution along the chosen space-filling curve scales perfectly when the number of processes for a fixed number of l-grids is increased and, thus, reduces the number of l-grids per process. At some point, all systems exhibit a levelling off for lower numbers of l-grids per process, which can be seen drastically in the case of the TUM MAC cluster at depth 5.

Figure 4.9(b) also shows a separation into mostly two bands: one containing the Blue Gene/P exchange times and one containing SuperMUC and TUM MAC cluster measurements. Comparing the SuperMUC to the Shaheen Blue Gene/P exchange times, it can be observed that SuperMUC is around 5.9 times faster for exchanging the same amount of data over the network interconnect (compared here exemplary for depth 6).

Altogether, it can be summarised that the chosen Lebesgue space-filling curve – together with the communication subroutines divided into three different phases – performs and scales (nearly) perfectly for different numbers of processes and different numbers of l-grids per



(a) time plotted versus the number of processes used for computing



(b) time plotted versus the number of l-grids per process

Figure 4.9.: Communication times for a total ghost layer exchange of 9 independent variables per cell in seconds on three different computing platforms for a 3D domain of fully refined l-grids with a refinement level (2,2,2) up to depth 8 and a d-grid size of (16,16,16).

process on several computing platforms, thus being perfectly suitable for our large-scale HPC purposes.

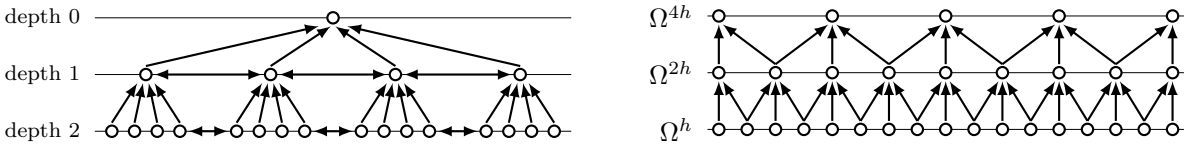
4.6. Multi-Grid-Like Solver

A different representation for the communication scheme presented in Section 4.3 with a recursive subdivision can be seen in Figure 4.10(a). In order not to overload the picture with arrows, only the bottom-up and the horizontal communication phases are depicted. The top-down communication is the same with vertical arrows pointing in the opposite directions in-between different depths and propagates the information downwards.

For a better comparison of the multi-grid approach with the communication structures, a few common terms should be established. In the l-grid hierarchy introduced in Section 3.8 and used by our communication structure, the coarsest l-grid is always referred to as the root l-grid having a depth of zero. Hence, the depth increases from top to bottom, meaning that the tree is actually 'growing downwards' as is customary in computer science (cf. [Bad13], or [Sam90]). The terms of the communication phases such as bottom-up and top-down also refer to this ordering. In multi-grid theory such as [BHM00] or [TOS01], usually the fine d-grids are depicted on top and the d-grids get coarser when descending this hierarchy.

For the remainder of this work, we define 'descending' as moving from the root to the most refined leaf nodes (also termed top-down) and 'ascending' as moving from the leaf nodes to the root node (also termed bottom-up), regardless of the depiction of the tree structure.

Thus, Figure 4.10(b) has also been adapted to represent the ordering chosen here.



(a) communication structures (b) full weighting

Figure 4.10.: Comparison of communication structures with a full-weighted multi-grid approach.

By comparing the communication procedures of Section 4.3 represented in Figure 4.10(a) to the multi-grid restriction shown in Figure 4.10(b), some similarities can be seen. Generally speaking, data on a finer d-grid is aggregated and sent upwards to parent d-grids in a recursive process during the bottom-up communication phase, until the root d-grid is reached. This is very similar to what happens in a geometric multi-grid solver. The approximate solution is smoothed a few times and restricted to a coarser d-grid, which will be repeated until the coarsest d-grid has been reached. In a standard geometric multi-grid approach, the solution is then obtained by means of a direct solver, for example, and an error correction is prolongated to finer d-grids in a repetitive process, until the finest d-grid has been updated. Something

similar happens during the top-down communication phase, where data is interpolated to finer d-grids until the most refined d-grids have been updated.

By comparing the similarities between the two approaches, the obvious thing is to integrate a geometric multi-grid solver approach using the already available communication patterns of the exchange mechanism. In the standard multi-grid case, coarse d-grid points are usually subsets of finer d-grids. This is not the case in a cell-centred approach, where the coarse d-grid points do not coincide with finer d-grid points. Thus, the data restriction and prolongation operator has to take account of this feature in the current concept and data needs to be interpolated to the respective points in space. Hence, in this work the solver is called 'multi-grid-like'.

As the geometry definition is prescribed on the finest d-grids, the question of choosing boundary conditions arises on the coarsened d-grids. [MST10] suggest a 'hierarchy of significance' in boundary conditions: Dirichlet conditions always take precedence over standard fluid cells, which themselves take precedence over Neumann conditions. This means that a coarse d-grid cell will be set to a Dirichlet boundary condition if one of its corresponding fine cells contains a Dirichlet condition. If there is no fine Dirichlet boundary condition, and there is at least one fluid cell, the coarse cell will be set to fluid. Only if there is no fluid cell at all (and obviously no Dirichlet cell), will a coarse cell be set to a Neumann boundary condition. This leads to 'disappearing' bubbles of Neumann boundary conditions in a fluid domain while coarsening the d-grids for a multi-grid approach, as shown in Figure 4.11.

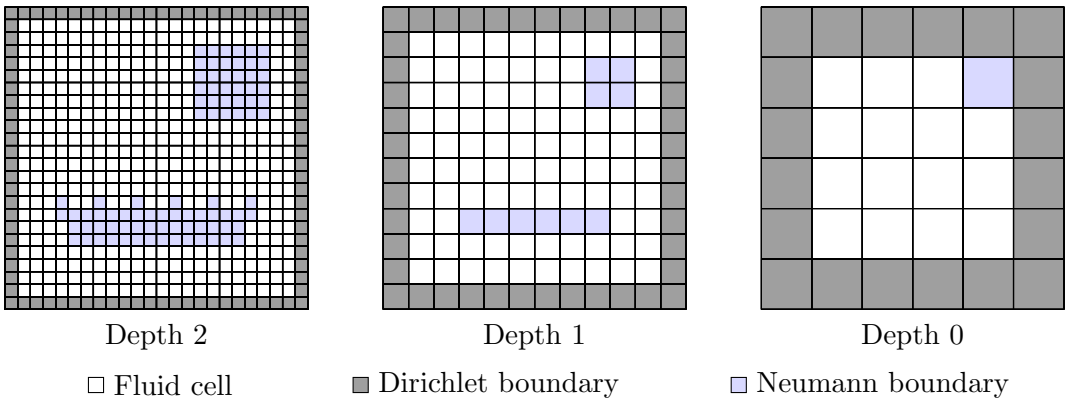


Figure 4.11.: Boundary condition propagation throughout the different d-grid levels.

The settings and determination of the boundaries in the various coarse d-grids is done once in a pre-processing step after the initial setting of the boundaries on the finest geometry level. Every time the geometry is changed in a computational steering approach, a 'propagation' of the boundary conditions over the different coarse d-grid levels has to be repeated, for example.

From the description of the treatment of the interior boundary conditions, it is obvious that the regions around boundaries can cause trouble in terms of convergence. In the worst case, a divergence of the solution was observed, depending on the number of l-grid levels and smoothing steps. Thus, in order to warn the user of such behaviour (mainly caused

by bad parameter settings), warnings were integrated into the code which will abort the computation. Another possibility would be to automatically detect and adapt the parameters which need fixing.

4.6.1. Convergence Tests for Poisson Equations

As the multi-grid-like solver is responsible for solving the pressure Poisson equation $\Delta p = f$ introduced by the fractional-step approach, the solver was first tested on a stand-alone basis for the given Poisson equation with prescribed boundary conditions. Therefore, Algorithm 3.1 has been adapted to the data structure using the exchange and interpolation mechanisms as restriction and prolongation operators. Line 2 and line 9 of Algorithm 3.1 dictate a relaxation of the residual equation α_1 and α_2 times. A Jacobi-based relaxation step with an under-relaxation (or dampening) of $\omega = 2/3$ is used as a relaxation process.

As the restriction operator used is closer to a pure injection than to a full-weighting operator, convergence problems can arise at some point. Tests showed that, depending on the depth and settings in the domain, convergence could not be reached for some settings of α. If α was enlarged, convergence could again be reached, with the drawback of increased communication costs. Thus, a study regarding different strategies for choosing α was undertaken.

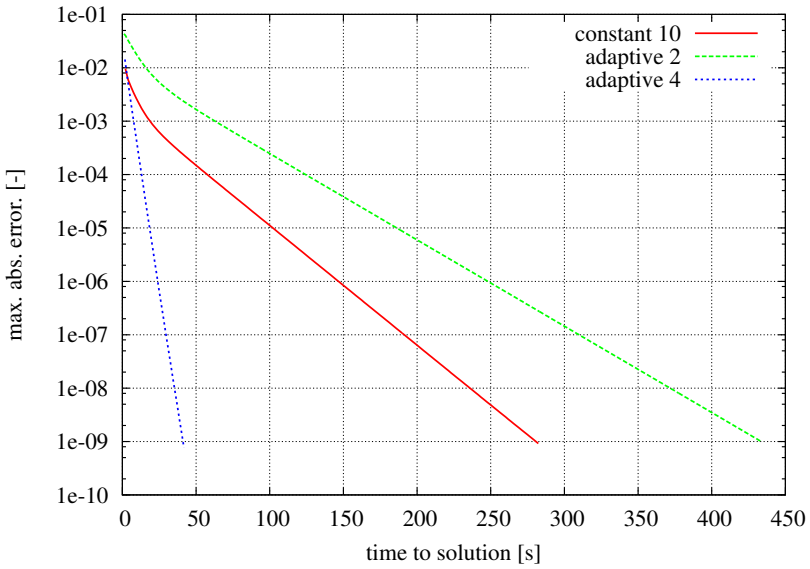


Figure 4.12.: Time to solution for different choices of $\alpha_{1,2}$ computed on 1,024 processes on Shaheen for a 3D Laplace problem on a uniformly refined l-grid with depth 6 using refinements (2,2,2) and d-grid sizes (4,4,4) resulting in a total of 299,593 l-grids and 19.17 million cells.

Figure 4.12 shows the time to solution in seconds for solving a 3D Laplacian problem on a cubic domain refined using bisections to depth 6 and a d-grid size of (4,4,4) on Shaheen for different smoothing strategies. While refining to depth 6 and using $\alpha = 4$, convergence could not be reached. For $\alpha = 10$, convergence could be achieved and the solution with an

accuracy of 10^{-9} was obtained after approx. 280 seconds or 150 multi-grid v-cycles (curve named *constant 10*).

If one thinks about the l-grid hierarchies, it is obvious that a data exchange involving finer l-grids is more expensive than exchanging data over coarse l-grids. Thus, one idea is to smooth fewer times on fine d-grids and more times on coarse d-grids. Hence, an adaptive strategy is applied: on the finest level at depth d, $\alpha = 2$ is chosen; at $d-1$, α is set to 4, at $d-2$, α is set to 8, etc. Hence, at each level, the alpha value is multiplied by a factor of two. Even if the communication costs are reduced (compared to *constant 10*), the convergence characteristics were less promising: curve *adaptive 2* delivered a solution with an accuracy to 10^{-9} only after 430 seconds or nearly 480 v-cycles. Although the system did not diverge, smoothing two times on the finest level is obviously not enough. If the initial smoothing is set to $\alpha = 4$, however, and subsequently multiplied by a factor of two in every level, an exceptional algorithmic speed-up can be observed. Curve *adaptive 4* achieves the result with an accuracy of 10^{-9} after approx. 40 seconds and needs only 27 internal v-cycles. As a bisection was used here, a multiplication with 2 for every level was chosen.

4.6.2. Performance Measurements of the Multi-Grid-Like Poisson Solver

Performance measurements were done in order to evaluate the scaling of the multi-grid-like solver for a Poisson equation. In all subsequent tests, an adaptive setting of the intermediate smoothing steps α was chosen, as this turned out to provide the best run times.

The chosen problem setup consists of a 3D cubic cavity of $1 \times 1 \times 1$ m, where the Laplacian problem

$$\Delta p(x,y,z) = \frac{\partial^2 p(x,y,z)}{\partial x^2} + \frac{\partial^2 p(x,y,z)}{\partial y^2} + \frac{\partial^2 p(x,y,z)}{\partial z^2} = 0 \tag{4.4}$$

for the pressure distribution p was analysed. As boundary conditions, fixed Dirichlet boundary conditions were applied on the east and west side according to

$$p(0,y,z) = p(1,y,z) = 1 \qquad \forall\ y,z \quad , \tag{4.5}$$

whereas all other sides were set to $p = 0$.

This setup corresponds to a certain degree to the basic internal equation solved in every iteration of the fractional step-based approach and, thus, its performance behaviour is crucial for the overall program execution time.

In order to analyse the impact of the chosen block size (i. e. d-grid size) on the time to solution behaviour, a study was carried out on the MAC cluster and the Blue Gene/P installed at UVT in Romania. All computations were executed using 256 processes and

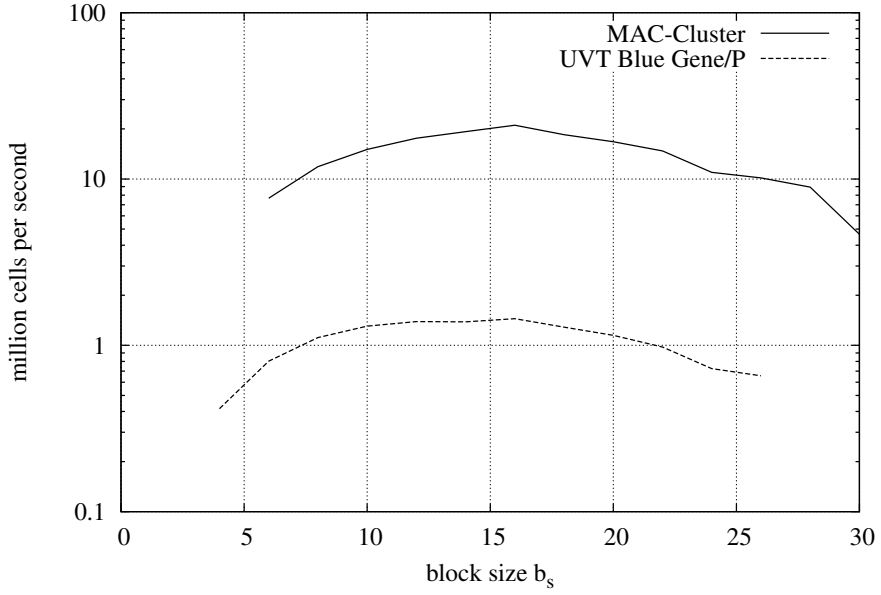


Figure 4.13.: Number of cells solved per second plotted over the chosen block size (b_s, b_s, b_s) (i. e. d-grid size) for a uniform l-grid setup using a recursive bisection up to depth 5 and computed using 256 processes on different systems.

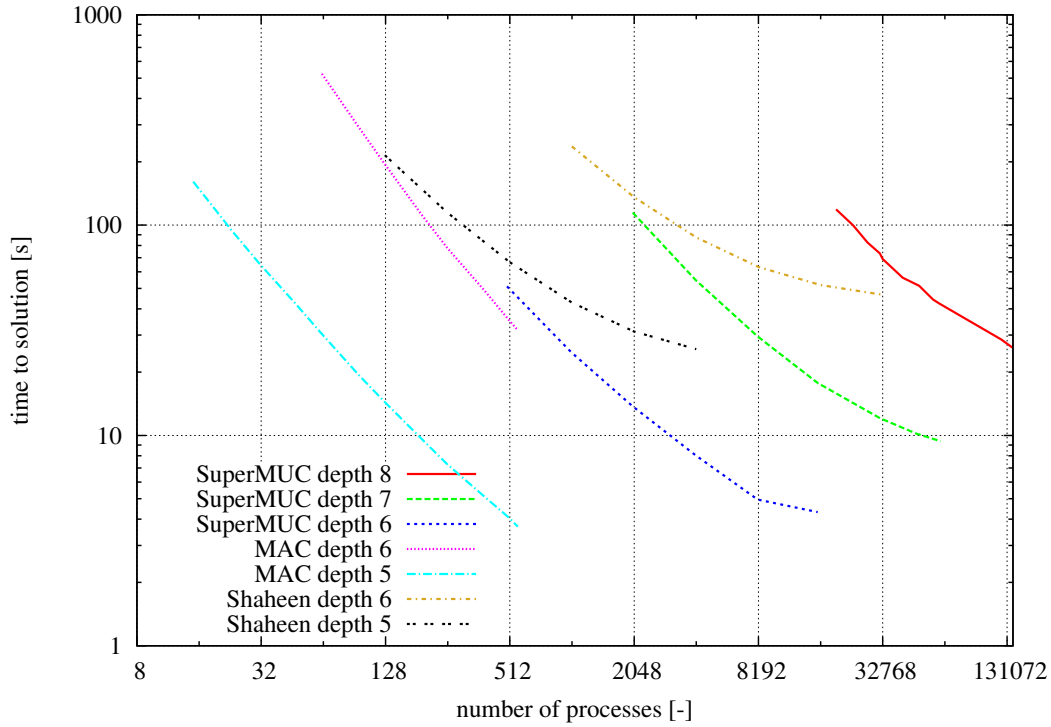
a uniformly refined setup on depth 5 having 37,449 l-grids in total[5] or approximately 146 l-grids per process. The block size was varied from around 4 to 30. As the number of cells and, thus, unknowns in the system changes with increasing block sizes, it is not possible to use the absolute time to solution for a comparison. Hence, the wall time was converted to the number of cells solved per second and plotted over the block size, as shown in Figure 4.13.

A maximum throughput in the analysed region is reached if the block size is chosen as (16,16,16) in the 3D setup. For the upcoming performance measurements these values were used.

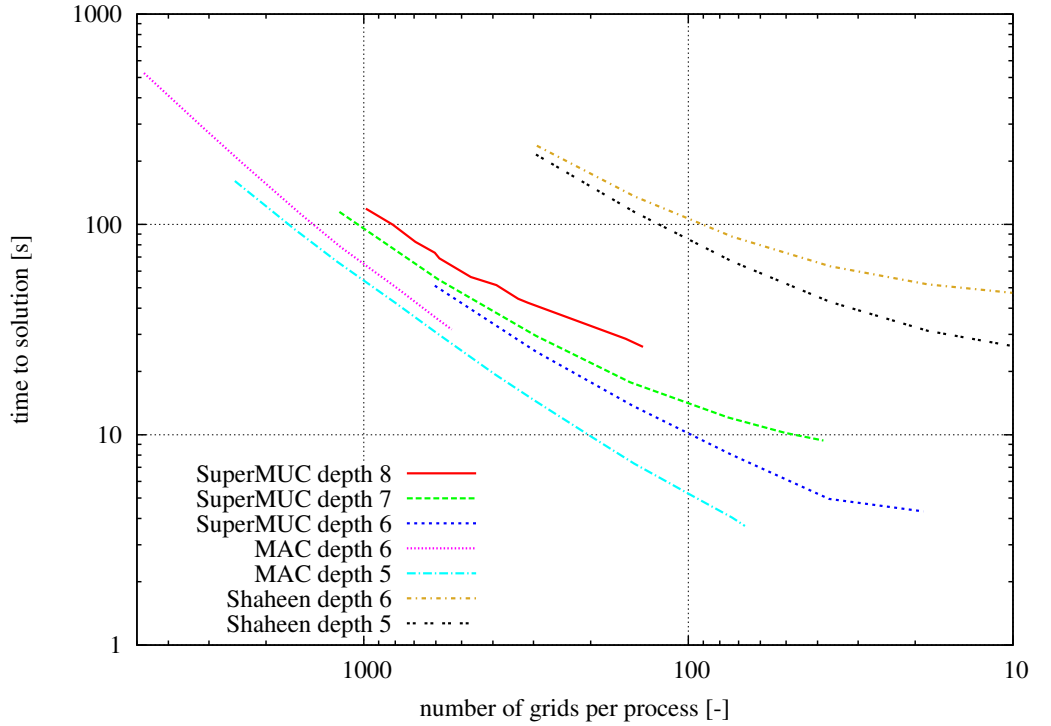
Furthermore, it can be observed that the MAC cluster and, thus, the Sandy-Bridge-based architecture is an order of magnitude faster than the PowerPC-based architecture of the Blue Gene/P. This can be explained by taking into account the difference in clock speed: the PowerPC processor has a clock speed of 850 MHz, whereas the Sandy-Bridge processor uses 2.7 GHz, which accounts for a factor of 3.2. Furthermore, the network speed of the MAC cluster is around 6 times faster than the Blue Gene/P network interconnect, which was determined according to Figure 4.9(b). Additionally, other effects such as different cache sizes or memory bandwidths have to be taken into consideration, so that in the end a factor of 10 can be justified from a hardware-based point of view.

Run time measurements of the implemented multi-grid-like Poisson solver were carried out,

[5]recursive bisection with 2^3 children for depth 5 results in $(2^3)^0 + (2^3)^1 + (2^3)^2 + (2^3)^3 + (2^3)^4 + (2^3)^5 =$ 37,449 l-grids



(a) time plotted versus the number of processes used for computing



(b) time plotted versus the number of l-grids per process

Figure 4.14.: Time to solution on three different computing platforms for a 3D domain of fully refined l-grids with a refinement level (2,2,2) up to depth 5, 6, 7, or 8 and a d-grid size of (16,16,16).

and times to solution in order to reach a predefined error accuracy of 10^{-10} are presented in Figure 4.14. For the measured domain setups, information about the number of l-grids and cells as well as the total memory requirement in gigabyte can be seen in Table 4.1.

Table 4.1.: Domain sizes for measurements using a recursive bisection and a block size of 16^3.

depth	number of l-grids	number of cells	tot. mem. req. [GB]
5	37,449	$0.15 \cdot 10^9$	54.7
6	299,593	$1.23 \cdot 10^9$	437.6
7	2,396,745	$9.82 \cdot 10^9$	3,500.9
8	19,173,961	$78.54 \cdot 10^9$	28,007.2

Depending on the system setup and the maximal available memory per process, not all combinations can be computed, thus limiting the minimal number of processes with which a given system can be executed. Except on the MAC cluster, all lower limits were nearly completely exhausted in terms of memory consumption. The maximal upper limit of processes is given by the system itself and by the number of l-grids per process, which cannot drop below one, as this would mean a waste of computing resources.

Figure 4.14(a) shows times to solution in seconds plotted over the number of processes up to a maximum of 139,008 on SuperMUC. It can be observed that the total run time is reduced by increasing the number of processes, which is quite desirable. If one focuses on the curve for SuperMUC and the MAC cluster on depth 6, it can be further observed that they are nearly similar. This is expected since, as mentioned before, the underlying computing system is exactly the same.

From Figure 4.14(a) a general observation can be made: all curves show the same tendency of levelling off when the number of processes increases. In the double logarithmic plot they behave almost linearly at first, but at some point they tend towards asymptotic behaviour at higher number of processes. This can be explained by the communication to computation ratio: the more one process has to compute, the better it can interleave the communication and the less overhead the system has. In this case, an increase in the number of processes will also have a large effect on the time to solution. If less work load per process is present, the ratio of communication to computation increases and produces more communication overhead between the computing nodes and, thus, an increase in the number of processes does not have a large influence any more. In the asymptotic limit, there is nearly no computation at all and, thus, no reduction of the time to solution can be expected.

Similar to the exchange time measurements, the time to solution can also be plotted over the number of l-grids per process, which is shown in Figure 4.14(b). It is obvious that the plot contains two groups, which consist of the times to solution for PowerPC-based systems and Sandy-Bridge-based systems. This plot highlights again the slower time to solution behaviour of the PowerPC-based system versus the Sandy-Bridge-based system of around one order of magnitude (for 100 l-grids per process).

In this plot the aforementioned tendency to level off can be seen very well, and it can be used as an indication of the number of l-grids per process it is still reasonable to use, and should not be decreased. As a rule of thumb (according to the presented measurements), it could be said that fewer then 100 l-grids per process are not reasonable any more if a recursive bisection and a d-grid size of 16^3 is used in a 3D setup.

A different comparison and evaluation can be done by computing the strong speedup. During a strong speedup measurement, only the number of processes is increased, while all other simulation parameters (especially the domain size) remain constant. Hence, with increasing number of processes, the work load per process reduces and a communication overhead is generated. Therefore, a levelling off is expected at some point in the curve.

As, for the desired number of l-grids, memory requirement restrictions meant it was not possible to perform a computation on one process only (cf. Table 4.1), a scaling to the lowest possible time to solution $T(p_{smallest})$ was done. Thus, the strong speedup using p processes is computed by

$$S(p) = \frac{T(p_{smallest})}{T(p)} \cdot p_{smallest} \quad . \tag{4.6}$$

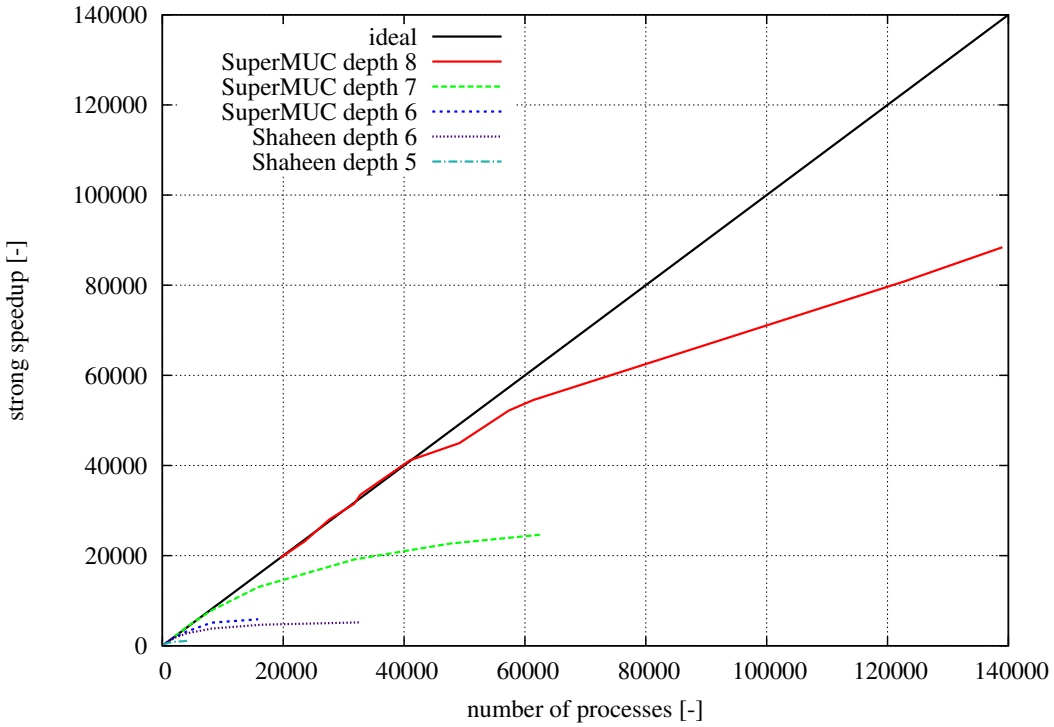
The results of the speedup measurements can be seen in Figure 4.15. Figure 4.15(a) shows the speedup on the complete SuperMUC thin island system for up to 139,009 computing processes and 1,024 NBH server processes. Furthermore, results for 50% of the size of Shaheen (i. e. 32,768 processes) are depicted. In order to better analyse the code's behaviour using lower numbers of processes, an excerpt (of the same measurements) up to 16,384 processes is shown in Figure 4.15(b).

It should be mentioned at this point that the run time measurements with the complete SuperMUC system were realised during the 'Extreme Scaling Workshop' held at LRZ in June 2014, in which the system was in special block operation mode. Thus, special arrangements were made to facilitate measurements and comparisons, by fixing the core clock frequency to 2.7 GHz instead of having a dynamic adaptation, for example.

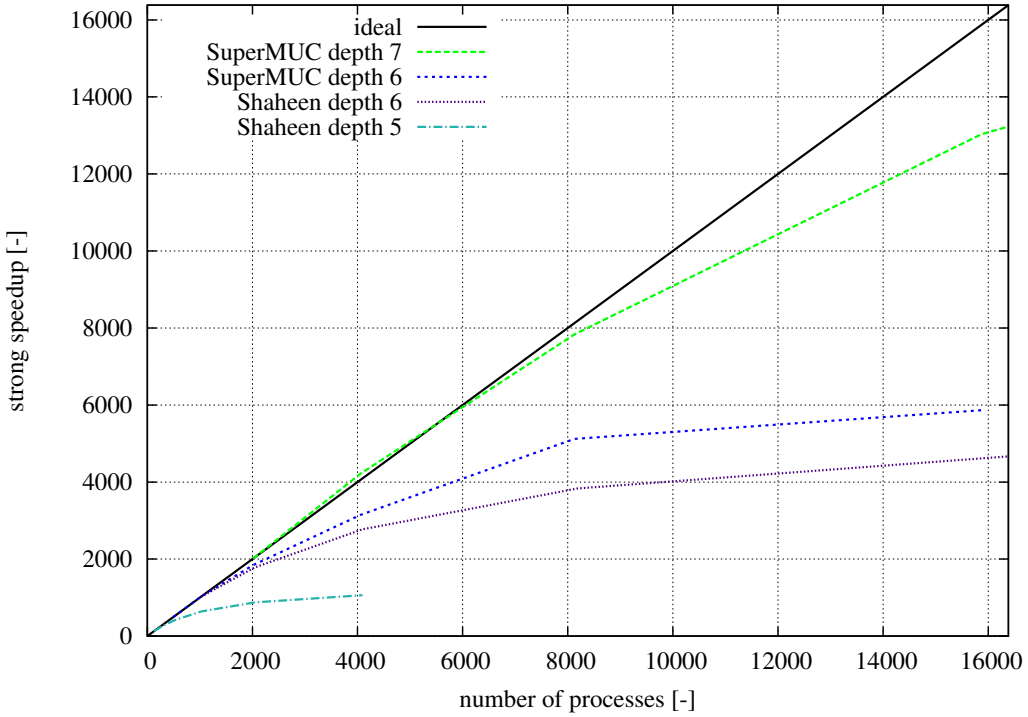
Especially if using more than 32,768 processes on SuperMUC, a levelling off for depth 7 can be seen, where the efficiency is around 60%. Obviously, depth 6 performs much worse and computations were only performed up to 16,384 cores on SuperMUC. Using the complete SuperMUC system and a problem setup at depth 8, an efficiency of 64% can still be reached in a strong scaling measurement.

On the other hand, if the excerpt in Figure 4.15(b) is regarded, a very similar behaviour for depth 6 on SuperMUC can be observed, where the efficiency at 8,192 processes is approx. 63% and at 16,384 processes approx. 37%. Furthermore, the efficiency of depth 7 on SuperMUC using 16,384 processes is at 82%, which is quite acceptable. This shows clearly that it is very important to have enough computational work load available per process, as otherwise the system is not using its resources in an optimal way, as the speedup results indicate.

It is also interesting to note that the general speedup behaviour for different depths on Shaheen and SuperMUC is not as different as the time to solution results presented in



(a) strong speedup from 1 to 139,008 processes (all)



(b) strong speedup from 1 to 16,384 processes (zoom)

Figure 4.15.: Strong speedup results on two supercomputing platforms for a 3D domain of fully refined l-grids with a refinement level (2,2,2) up to depth 5, 6, 7, or 8 and a d-grid size of (16,16,16).

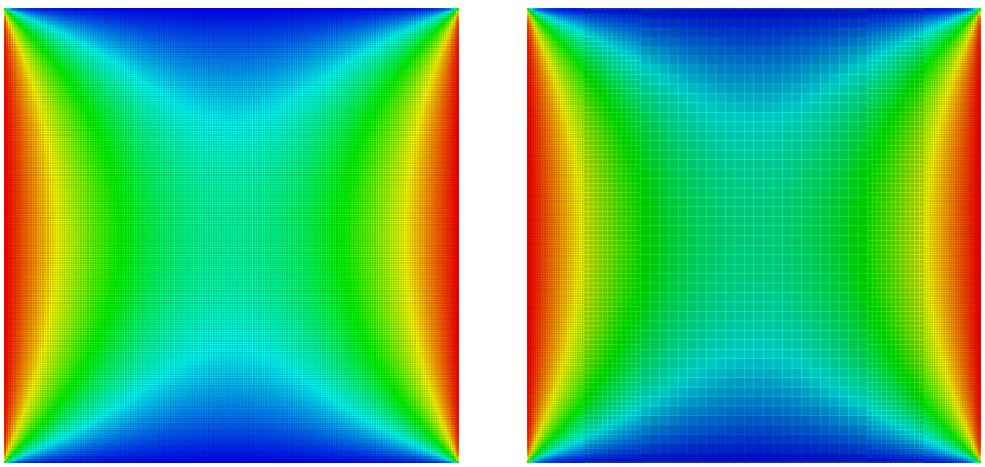
Figure 4.14(b). Depth 6 on SuperMUC has a better strong speedup behaviour than on Shaheen, which can be explained in combination with Figure 4.9(b) which shows the ghost cell exchange time behaviour.

It should be mentioned here also that the speedup values are computed in order to get an idea of the code's performance while scaling the number of processes upwards. The optimal solution will, however, always have a minimal time to solution, regardless of the speedup values. Nevertheless, they can indicate that there is still optimisation potential in order to increase the performance even further.

As a conclusion regarding the performance analysis, it can be summarised that the performance of the code is very dependent on the available hardware architecture (involving both computing and network resources) and the tuning of the parameter settings to this specific architecture. Therefore, it is imperative to run small tests before the execution of large, long-running examples in order to find optimal settings for different parameters which are not yet fixed by physics or the problem itself.

4.6.3. Error Assessment of Adaptive vs. Uniform Grid Setups

The data structure layout is designed to allow the generation of adaptive setups depending on refinements in the l-grid hierarchy. Until now, all measurements were done for uniformly refined l-grids up to a given level. In order to determine the accuracy in terms of errors with respect to a uniform solution, comparisons and measurements were presented in [FMR14] and will be briefly summarised here.



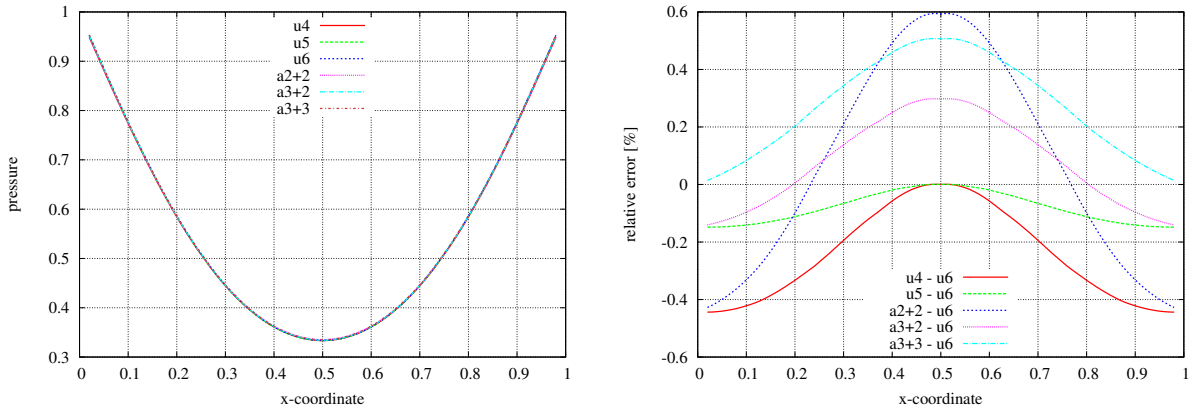
(a) uniform setup at depth 4 ($u4$) (b) adaptive setup at depth 4 ($a2+2$)

Figure 4.16.: 2D cut through a 3D test problem for testing the adaptive code (similar l-grid construction to Figure 4.7).

The problem setup for the error assessment of the adaptive behaviour is the same one as stated in the previous performance measurement section. A 2D cut through the middle

of the 3D domain is shown in Figure 4.16. The naming conventions can be interpreted as follows: $u4$ is a uniform refinement up to depth 4, $a2+2$ is an adaptive l-grid setup, which uses a uniform refinement up to level 2 and twice a subsequent adaptive refinement towards the east and west edges of the domain. This gives the adaptive setup the same resolution at the boundary of the domain as the uniform one. The computations were performed either on Shaheen or on the UVT Blue Gene/P, and the d-grid size was selected as (4,4,4) with a refinement of (2,2,2).

A smaller d-grid size was selected here in order to generate higher depths with a lower number of cells in order to evaluate a worst case scenario, as the focus in this section is more on the error assessment than on the run time performance.



(a) Pressure distribution along the cutline (b) Relative error in % along the cutline

Figure 4.17.: Data along the cutline in x-direction through the domain at $y = z = 0.5$.

A first glance at Figures 4.16(a) and 4.16(b) shows similar results. In order to characterise the error in the system, a cut along the x-axis was introduced at $y = z = 0.5$ for different domain refinements. Figure 4.17(a) shows pressure values along the given x-axis cutline. They already coincide quite well. In order to further quantify the accuracy, a detailed error assessment was performed by computing relative point-wise errors to a reference solution which was obtained by an 'overkill' solution using a uniform refinement at depth 6 ($u6$) according to

$$e_{rel}(x) = \frac{p(x) - p_{u6}(x)}{p_{u6}(x)} \cdot 100 \ [\%] \quad . \tag{4.7}$$

The results of the error computation are plotted in Figure 4.17(b). Two different error characteristics can be observed in this plot: the uniform $u4$ and $u5$ with respect to $u6$ as reference have a higher error towards the boundaries and nearly no error in the middle of the domain. This can be explained by the higher gradient towards the boundaries which are not approximated as well when using larger cells. A maximal error of 0.45% can be observed for the uniform refinement up to level 4 and 0.15% for level 5.

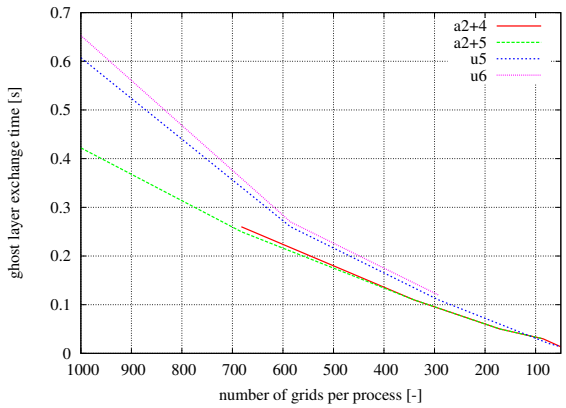
In the adaptive case, however, the error distribution looks a little different. For all discretisations, the highest error of around 0.6% can be observed in the middle part, where

the refinement is coarse. Towards the edges, the error reduces to the same level as in the corresponding uniform reference case.

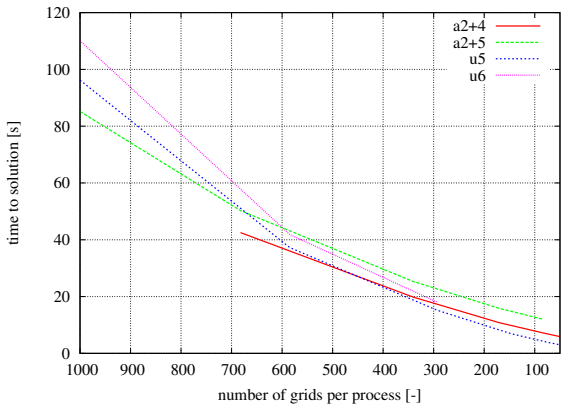
Summarising, it can be said that the adaptive setup produces results whose accuracy is acceptable if all factors are taken into account. The next question which arises concerns the performance of the adaptive l-grid setup, which is discussed in the following section.

4.6.4. Performance Measurements of the Adaptive Grid Setup

A fair comparison in terms of processes is quite difficult, as different l-grid distributions will have a different load per process. Therefore, the performance analysis results are displayed in a different manner. In order to compare similar load patterns, the exchange time as well as the time to solution is plotted over the number of l-grids per process.



(a) Data exchange time in seconds plotted over the number of l-grids per process



(b) time to solution plotted over the number of l-grids per process

Figure 4.18.: Scaling and performance measurements on the UVT Blue Gene/P using up to 4,096 processes. A refinement of (2,2,2) with a d-grid size of (4,4,4) was chosen.

In general, it can be observed that a higher number of l-grids will produce a higher exchange time as well as a higher time to solution, which is not unexpected. It can also be seen from the diagrams in Figure 4.18 that, for the given load distribution from Section 4.5 according to the Lebesgue space-filling curve, the run time as well as the exchange time behaviour is very similar to the uniform case.

Hence, it can be concluded that using an adaptive l-grid structure will not introduce a large overhead in the l-grid hierarchy compared to a uniform refinement to a certain depth.

The diagrams in Figure 4.18 show a similar characteristic to the previously measured results in Figure 4.9(b) and Figure 4.14(b), with the exception that a different d-grid size was used, and consequently more ld-grids could fit into one process without running into memory restriction limits.

Table 4.2.: Bitwise encoding of boundary conditions in an 8 bit unsigned character.

Bit position	7	6	5	4	3	2	1	0
Description	Geometry marked solid	*currently unused*	*currently unused*	Outflow	Temperature BC	Pressure BC	Velocity BC	Type of Cell
Bit value 0	No			No	Fixed	Dirichlet	Noslip	Fluid
Bit value 1	Yes			Yes	Flux	Neumann	Slip	Solid
Bit mask	0x80	0x40	0x20	0x10	0x08	0x04	0x02	0x01

4.7. Boundary Condition Encoding

The boundary conditions are encoded bit-wise in an unsigned character stored in every cell. The current configuration can be seen in Table 4.2. The type of cell in bit position 0 can be determined by the voxelisation process described in Section 4.2. Using a combination of these flags at certain bit positions, a given boundary condition can be fully determined according to the given usage: boundary condition 25, written in binary as 00011001 (padded to 8 positions), for example, represents a solid modelling an outflow with a flux boundary condition in temperature, a fixed boundary condition for the pressure, and a no-slip condition for the velocity. Using 'binary and', 'binary or', and bit shifting operations, the given boundary condition can be extracted quite fast from the bit code.

4.8. Immersed Boundary Conditions

When applying standard boundary conditions on complete cells, the geometry description is a summation of blocks, and a stepwise approximation is the best possible realisation. When using adaptive refinements around boundary regions, the effects of such a stepwise approximation can be minimised, but they introduce numerical errors which have to be treated. From a physical point of view, these stepwise approximations create a higher surface roughness, which has an influence on the fluid flow's boundary layer thickness and composition. Therefore, a better treatment has to be introduced in future.

One possible solution is the so-called *immersed boundary method*. Work on this topic has been done by [FVOMY00], [KKC01], or [TF03], for example. By applying this method, it is possible to impose boundary conditions on geometric points which do not coincide with the computational d-grid and the corresponding cells, but still execute the main part of the computation on a regular d-grid and, thus, still have all the advantages mentioned in the regular finite volume decomposition but enhance the boundary condition treatment to a higher order as well as increasing the numerical accuracy [FVOMY00].

A similar concept called 'finite cell method' (FCM) was introduced in [PDR07] for problems in solid mechanics, but it can be generalised to other problems, too. FCM is an extension of higher order finite element methods using a fictitious domain approach. A standard finite element mesh can be embedded into structured or unstructured cells where classical h- or p-Ansatz functions are defined. This method shows very good results in terms of convergence and accuracy even in the presence of singularities (cf. [PDR07]). Hence, general ideas of this method could also be applied to the approach presented in order to exploit the benefits of the boundary condition treatment.

These methods are currently not implemented in the present code and were never part of this thesis. They shall be mentioned here as an outlook in order to improve the numerical accuracy of the results by eliminating the artificial roughness of a step-wise approximated surface. For now, an adaptive refinement towards the boundaries can tackle this problem, even if the maximal possible time step size is reduced by increasing the level of refinements.

4.9. Short Summary

This chapter described all the necessary implementation steps, procedures, and dangers which exist when creating a massive parallel Navier-Stokes-based simulation code, which can run on modern supercomputers, and solve engineering-based problems of daily life such as indoor air temperature distributions in occupied spaces.

A complete code with special focus on parallel computing was designed from scratch. Methods for domain generation such as a space-tree-based ld-grid generation from a given surface boundary description of a geometry were integrated. Furthermore, a unique addressing of cells and l-grids over different processes was implemented, which is essential in order to create the communication exchange functions which can transfer ghost halos in a possibly highly adaptive l-grid setup. The steps of the communication schemes were discussed and elaborated, and a neighbourhood server-based structure was introduced in order to keep track of all l-grids during run time without any huge memory overhead. Neighbouring search and bottleneck avoidance was discussed and performance measurements were given in order to illustrate the behaviour of multiple neighbourhood servers in a system. Load balancing strategies and ld-grid distributions over different processes were introduced and performance measurements were shown for different l-grid and d-grid setups. Run time adaptive ld-grid migration procedures were integrated that enable the migration of ld-grids from process to

process, in case a dynamic rescheduling for load balancing is necessary. A deeply interwoven multi-grid-like solver concept for elliptic partial differential equations such as the pressure Poisson equation was specially designed and tuned to interact with the proposed data structure, and fit extraordinarily well together. Run time measurements for uniform as well as adaptive l-grid setups were presented for several state of the art supercomputer systems including SuperMUC, the world's number four in terms of performance in June 2012. Furthermore, an error assessment for adaptive l-grid distributions was presented in order to quantify the error introduced by applying an adaptive setup which can save computing time and resources while computing with nearly the same accuracy as a uniform l-grid setup.

In summary, this chapter presents all the necessary steps and possible pitfalls in order to create a massive parallel computation framework with an interchangeable numerical kernel currently based on the Navier-Stokes equations.

5. Visualisation Concepts

A high-performance visualisation of the simulation results is of extreme importance in order to interpret the huge amount of data which results in an efficient way.

Basically, two different techniques can be distinguished: a pure post-processing visualisation approach, where simulation data is dumped to the hard-disk, being available for later purposes, and a run time visualisation, where data can be visualised on the fly during the course of a simulation. This section briefly introduces both techniques and focuses on the run time visualisation, which can again be integrated neatly into the data structure applied.

5.1. Post-Processing Visualisation Concept

The post-processing visualisation approach is quite straightforward. At user-specific, predefined visualisation steps, simulation results are dumped to the hard disk in a format suitable for later visualisation.

The visualisation for this work is done using ParaView[1], which is an open-source multiplatform data analysis and visualisation application designed for huge data sets. It is built on top of the visualisation toolkit library VTK[2] with a QT front-end.

VTK's native data format is a so-called VTK text file which can be either in ASCII or binary format and contains a complete description of the geometry in a boundary representation form, as well as all cell or point attributes.

Currently, each process evaluates all l-grids it is responsible for and writes all non-refined d-grids into a file. Hence, one visualisation time step always contains n files, where each file has its own part of the domain. These geometry parts are always non-overlapping and constitute the complete computational domain. At the end of the write-out step, the n vtk files are combined by a function out of the VTK libraries resulting in one single file per time step. The temporary n data files are deleted after the merging procedure.

This approach has advantages and disadvantages. An advantage is that no process is necessary for collecting all the data from the neighbouring nodes for a later write-out, which can be very time and memory consuming. In fact, for larger examples, this entails memory requirements of hundreds and hundreds of gigabytes of main memory for one process, which

[1] ParaView: `http://paraview.org`
[2] VTK: `http://www.vtk.org`

is usually not provided on supercomputers. A disadvantage is that the write-out phase puts quite a lot of stress on the network-based file system, as n processes try to access the same network location. Using an efficient parallel file system such as IBM's General Parallel File System (GPFS) on the Blue Gene/P system, this problem can be partially avoided.

Nevertheless, very large data files of multiple gigabytes might be the result of such a post-processing visualisation step and even a specially designed visualisation program such as ParaView will have difficulties dealing with this large data set, as reading from the disk will become a bottleneck at some point. Furthermore, the data might lie remotely on some supercomputer and it would first be necessary to transfer it completely to a local storage before using a standard visualisation tool. Therefore, an improved concept needs to be implemented in order to speed up the visualisation process and to efficiently handle very large data sets.

Today's 'buzzword' describing this problem is *Big Data* and refers to the fact that data sets are getting so extremely large and complex that they cannot be visualised completely without clever algorithms such as pre-selection, for example. [WSJ+12] states the top 10 challenges in working with big data and current developments in terms of interactive exploration.

In order to overcome the aforementioned drawbacks of file writing and off-line visualisation, one type of interactive visual exploration, namely a so-called *sliding window approach*, was implemented and is described in the next section.

5.2. Sliding Window Visualisation Concept

The *sliding window* visualisation concept can be used in order to bridge the gap between large data sets and finite bandwidths for data transfer from a computational back-end to a visualisation front-end.

Before diving into technical details of the given implementation, a few essential facts should be stated. If very large domains are to be visualised, usually two views are highly interesting: a global view, where a total overview over the entire domain can be seen, and a detailed view where a single detail is highlighted using a very high resolution on a small part of the entire domain. In other words, the underlying idea of the sliding window visualisation concept is to keep a constant bandwidth for transferring information content.

The following passages are excerpted and slightly adapted from [MFR13] and give an overview of the method. The sliding window concept limits the data transfer necessary between back-end and front-end by selecting only subsets of the computed results available. Hence, any user should have the possibility to choose both a window, i. e. a desired region of the computational domain, and a desired data density for the particular data to be visualised on the front-end. This window can be slid over the computational domain as well as increased or decreased in size, always determining which subset of data should be selected for the transfer. Whereas the window itself acts as a bounding box for the selection, the density defines the

number of data points inside that region to be considered for the visual display. The key feature of the entire approach is to keep the density constant while the window is being moved or resized, thus at any time the same amount of data is transmitted from the back-end to the front-end. This allows an optimal exploitation of the underlying network without exceeding any bandwidth limitations or causing unnecessarily long transmission delays that would harm the experience of authentic interactive computing.

In the case of high-resolution data, a full window covering the entire computational domain would force the back-end (depending on the chosen density) to skip lots of data points for the data transfer by selecting only every fifth or tenth data point in every dimension, for example. Hence, the user would obtain a qualitative representation without too many details which more or less makes it possible to catch global effects of the running simulation. When resizing the window, i. e. making it smaller, a smaller region of interest is covered and, thus, in order to keep the density constant, fewer data points should be skipped, providing more and more details for the visual display. The window size can be further decreased until the resolution of the computational domain is met, i. e. now each single data point inside the selected region is shown on the visualisation front-end. Obviously, the critical questions are how to select the right data points (cf. density) and, in case of a parallel simulation, how to select the right processes (cf. window).

The next section shows that the data structure presented in 3.8 favours the sliding window concept, as the hierarchy of l-grids together with their update scheme means an implementation of this concept is straightforward and questions concerning the correct selection of data points and processes are very easy to answer.

5.2.1. Implementation

The implementation of the sliding window concept is based on two major components, namely a server-side collector, responsible for collecting and collating the requested information on the back-end, and a client-side visualisation and interpreter tool at the front-end for sending data requests and receiving corresponding data streams from the back-end. Figure 5.1 shows a schematic representation of the complete structure.

5.2.1.1. Server-Side Collector

In order to keep the interaction between the data collection process and the actual computation processes to a minimum, a special dedicated collector process was created. The main structure of the collector consists of an infinite loop listening at a TCP socket on a dedicated port for a single character describing the next task for execution (arrow number 1 in Figure 5.1). One possible command from the client-side to the server-side would be to request a sliding window visualisation.

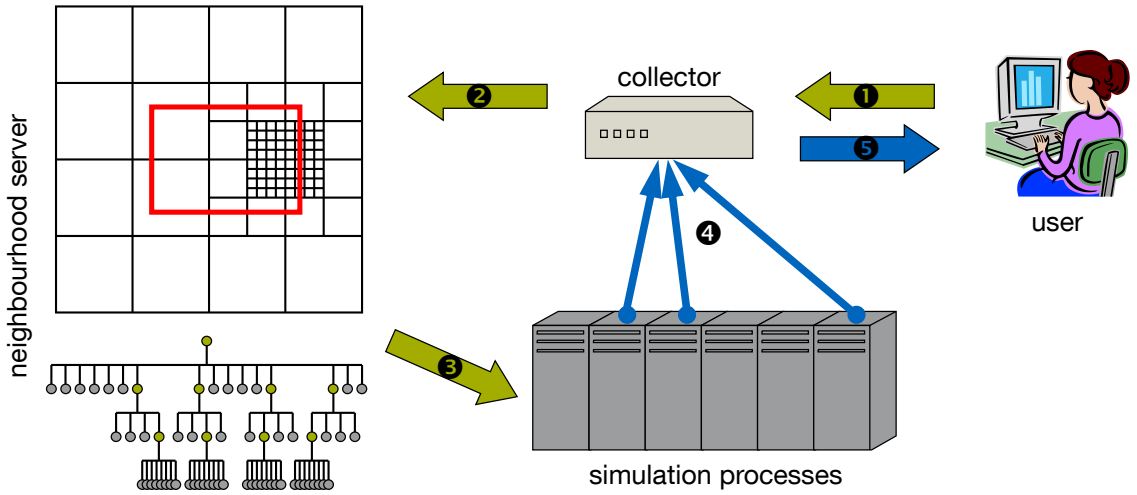


Figure 5.1.: Sliding window visualisation concept.

Once the client has sent the task initialisation, it will continue sending the necessary information, such as visualisation data type, maximum number of cells, and visualisation bounding box, via the socket connection to the collector process. The collector receives this information and submits a query to the neighbourhood server (arrow number 2), which has a complete topological and geometric overview of the stored l-grids. For every l-grid, the neighbourhood server performs an intersection test along the x, y, and z axes of the l-grid's bounding box using the transmitted sliding window bounding box. If the l-grid touches or lies completely inside the sliding window box, it is marked and added to the treatment list. If it is completely outside, the l-grid and all its l-grid children are ignored henceforth. The traversing order is given by the same space-filling curve used for load distribution in Section 4.5. Thus, the l-grid identification is executed from the root l-grid to the refined l-grids in deeper levels.

Once the maximum number of cells to display is reached and the relevant l-grids have been identified, the neighbourhood server sends orders to the corresponding processes containing the l-grid IDs (arrow number 3). The local simulation processes interpret the orders and send a data stream to the collector process (arrow number 4) using MPI messages on the very fast, dedicated cluster interconnect. As soon as a simulation process has sent the requested data, it can resume its normal operational mode and does not have any further interaction with the client. The collector reforms and compresses the data received in a way that it can be sent over a (possibly) slower TCP connection to the client (arrow number 5). As the collector is decoupled from the simulation processes and uses a separate MPI communicator, a slow client-collector connection has no effect on the simulation process itself, but only on the maximal achievable frame rate between client and server.

Instead of posting a visualisation task, the client can also interactively modify settings on the back-end, such as ordering l-grid refinements or changing boundary conditions and cell types. In this way, an interaction with the code during runtime is possible and at the same moment an interactive visualisation makes it possible to switch through different scales from

very large to very fine, whereas computations are always performed on the finest possible scale.

5.2.1.2. Client-Side Visualisation

One possible client-side visualisation is realised using a ParaView plug-in created alongside the main code and wrapped around a connector kernel, which handles the connection as well as the compression and decompression of the data streams.

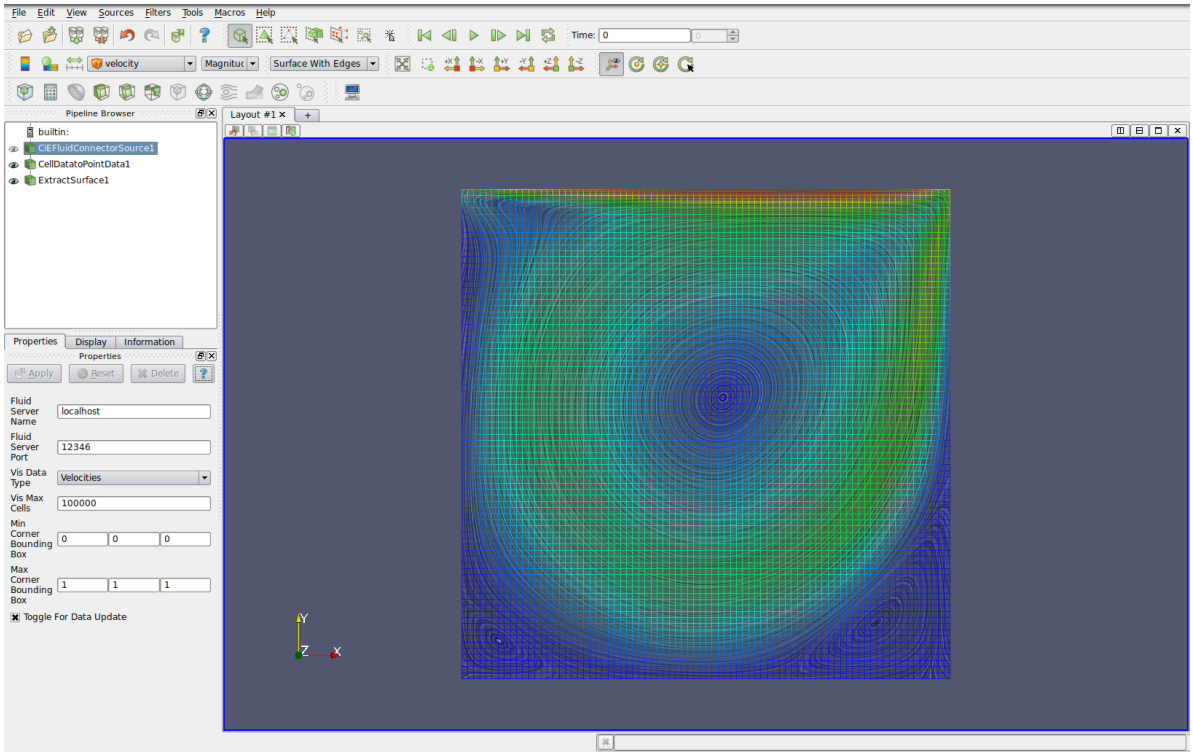
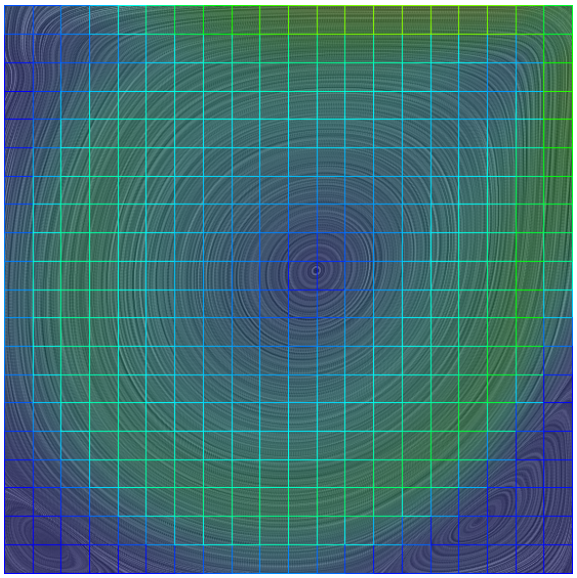
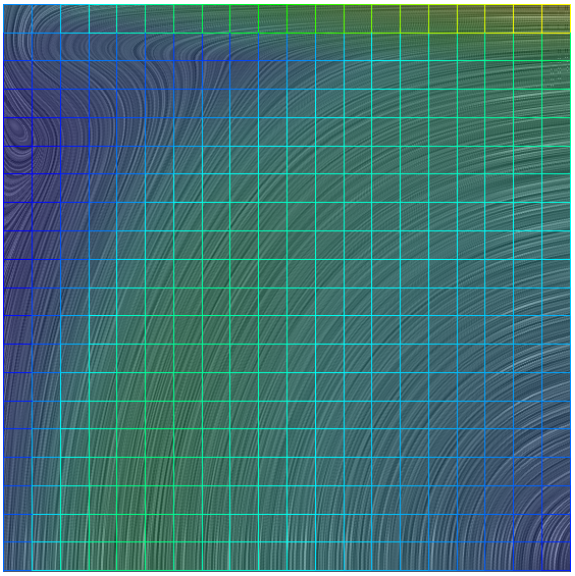


Figure 5.2.: ParaView as visualisation front-end enriched by a plug-in for the direct connection to the simulation results computed on the back-end (cluster or supercomputer) [MFR13].

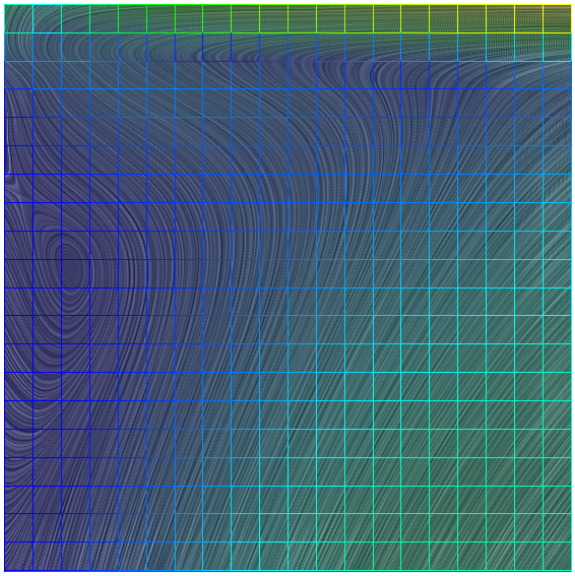
Figure 5.2 shows a screen shot of the ParaView GUI with the plug-in details on the lower left side. The user can enter the fluid server name and port as well as the data type to be transmitted, the maximal number of cells to display, and the bounding box coordinates of the sliding window visualisation. The plug-in connects to the collector process via TCP sockets and checks several prerequisites, such as the endianness and size of standard data types. If all prerequisites match, a connection can be established and among other tasks a visualisation can be requested (as described in the previous section). The bounding box coordinates, as well as the visualisation data type and number of cells to display, are sent to the back-end, which queries internally for the right data and delivers the results back to the front-end for visualisation. Internally, the connector kernel receives a long data stream which is decoded and stored in an internal VTK unstructured grid data structure which can then be immediately visualised using ParaView.



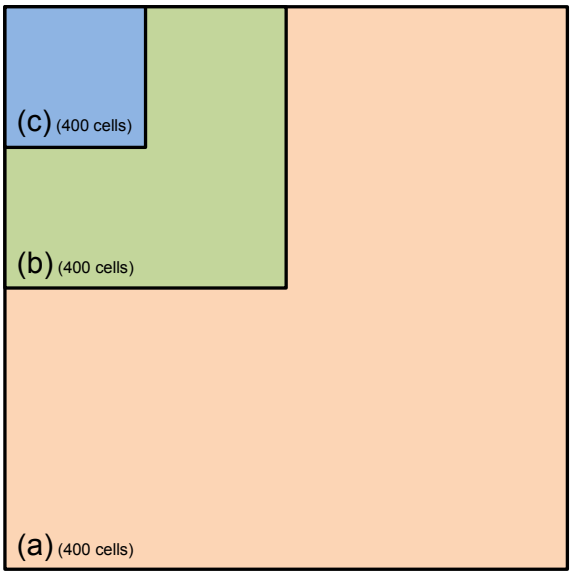
(a) Display of max. 400 cells in the sliding window bounding box (0;0) to (1;1)



(b) Display of max. 400 cells in the sliding window bounding box (0;0.5) to (0.5;1)



(c) Display of max. 400 cells in the sliding window bounding box (0;0.75) to (0.25;1)



(d) Different positions of the selections from a, b, and c

Figure 5.3.: Flow in a lid driven cavity for Re = 3,200 (represented using LIC – Line Integral Convolution [CL93]) visualised by the sliding window concept for different selections, always using a maximum of 400 cells (but always computed on the finest resolution of the computational domain). The vertex in the top left corner cannot be seen in the coarsest visualisation, only after zooming in. [MFR13]

Figure 5.3 shows an example of an interactive exploration of a lid-driven cavity. Specific details of this benchmark computation are presented and discussed in Section 6.1, but are not relevant at the moment.

In this example, the bounding box of the sliding window is decreased gradually, while the number of transferred cells is kept constant at 400. In general, the process of coarsening is non-continuous as d-grids can only be selected as a whole. Nevertheless, with the right choice of refinement parameters a nearly continuous coarsening could be realised. Figure 5.3(a) depicts the complete domain in a quite coarse resolution. The basic features of the flow can be seen, even if only a rough picture of some details is visible. While in Figure 5.3(a) no vortex can be observed in the upper left corner, it becomes clearer in Figure 5.3(b) that there is some more information available, while in Figure 5.3(c) the vortex can be seen clearly. Figure 5.3(d) shows the bounding box selection of the figures previously introduced. The colouring of the cells represents the velocity magnitude field, and the underlying image is a visualisation of the velocity vector field using the line integral convolution (LIC) technique [CL93].

5.2.2. Interactive Data Exploration on Large Visualisation Components

In the previous section, the sliding window visualisation concept was introduced and it was demonstrated using a small-scale example visualised on a standard workstation.

In order to demonstrate that this system is capable of powering much larger visualisation components, large-scale tests were performed in the visualisation core lab at the King Abdullah University of Science and Technology (KAUST) in the Kingdom of Saudi Arabia, where a large number of high-performance visualisation components are available from a 3D immersive, six-sided CAVE system called CORNEA, via the so-called NexCAVE, which is a scalable modular 3D environment of 21 tiled displays arranged on parts of a hemisphere, to a 40 tiled wall display system called AESOP (cf. Appendix B.3.1). Further information about the installed systems can be found on the respective website[3].

The visualisation tests were performed on the AESOP system and pictures of the tests are shown in Figure 5.5 and 5.6. Figure 5.4 outlines the basic setup in order to do a parallel visualisation using ParaView. Basically, the simulation code runs on a back-end computation cluster and a front-end visualisation ParaView client connects itself to the collector node on the cluster using the TCP socket communication protocol. The ParaView connector client runs on the front-end of the AESOP system, which itself is powered by a 10 node cluster, each node having four graphic cards connected to a total of 40 tiled displays.

The parallel visualisation itself is not realised by the author's own solution, but by using ParaView's internal client-server-concept for creating multiple parallel visualisation views

[3]Source: `http://www.vis.kaust.edu.sa`

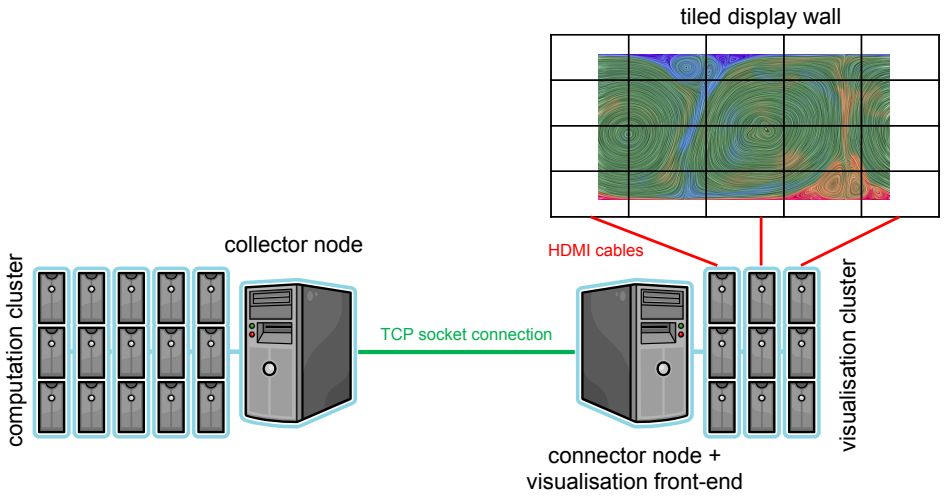
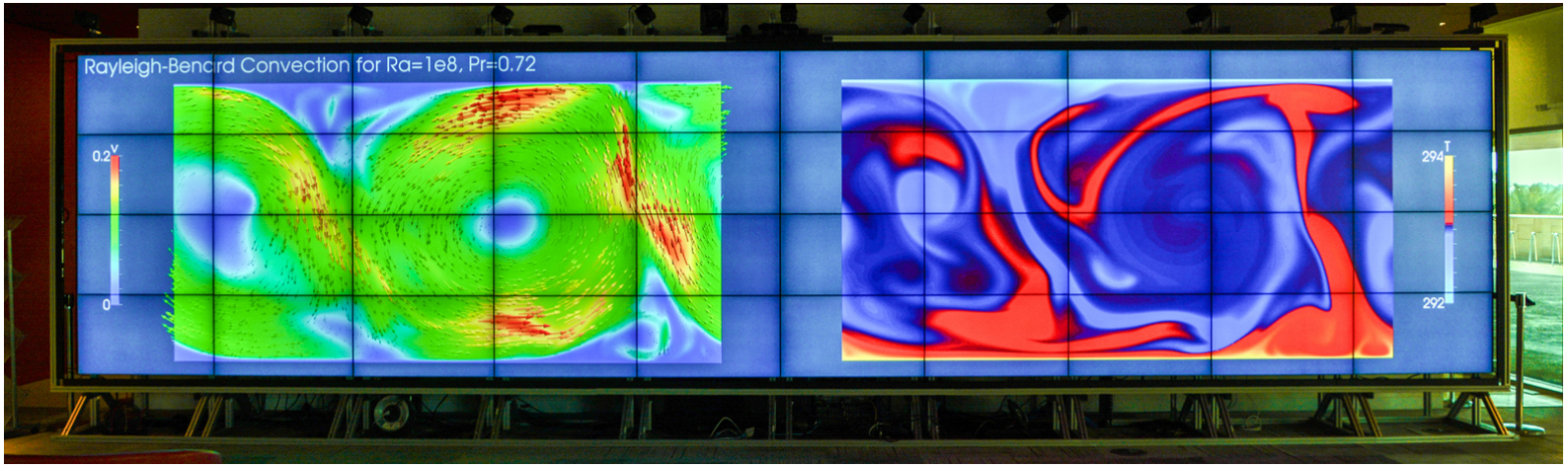


Figure 5.4.: Conceptual design of the simulation and visualisation connection.

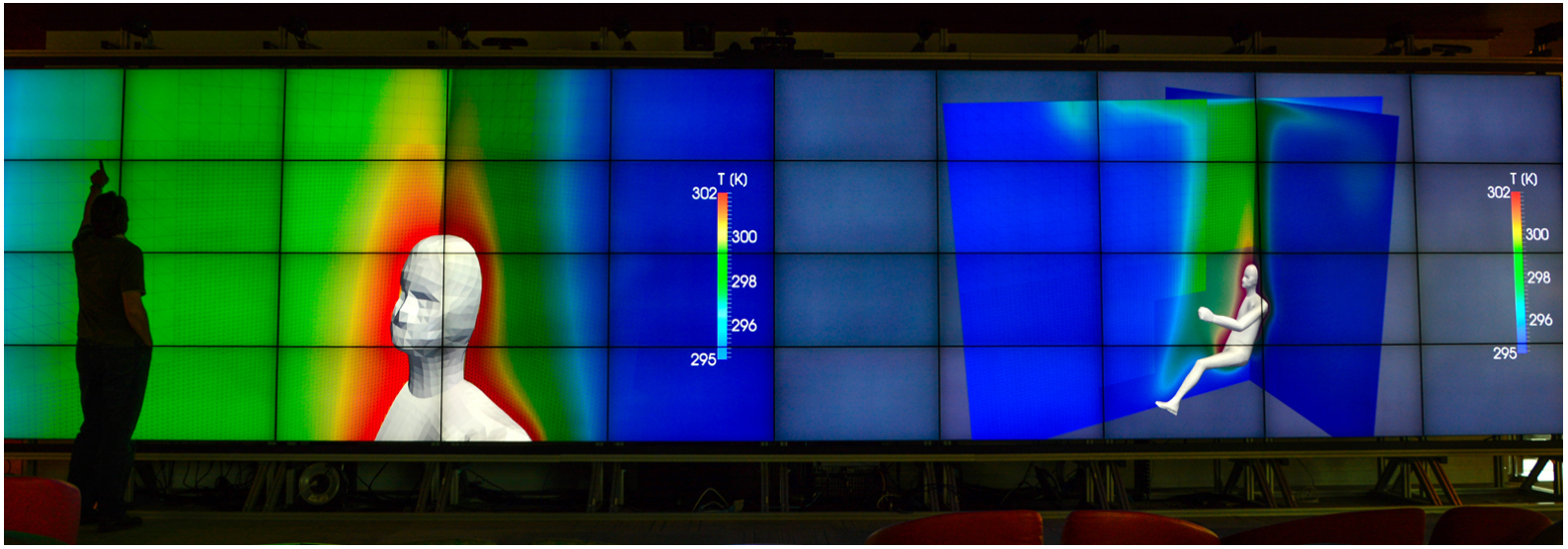
distributed over a visualisation cluster, which became quite powerful since the introduction of version 4.1.0.

While the simulation is running, the user connects to the collector node and can order ParaView to display the result on the AESOP wall. The user can then interactively change the bounding box of the sliding window and zoom into parts they want to examine in detail. This can be seen in Figure 5.5(b) which shows a heated manikin in a cold room, simulating a natural convection scenario. When the full view is selected, a lower resolution is transferred from the connector to the client in order not to exceed a predefined maximal bandwidth. If the user want to examine the manikin in a higher resolution, they order the server to decrease the sliding window's bounding box and, thus, automatically increase the number of details it is possible to see. Figure 5.6 shows such a detail on the AESOP wall, where the manikin display is switched to wire frame mode and details are highlighted.

To summarise, it is possible to say that the created data structure introduced in Chapter 4 is advantageous for this type of visualisation and interactive data exploration as well as for integrating a computational steering approach into future projects.



(a) Visualisation of a Rayleigh-Bénard convection problem with $Ra = 10^8$ and $Pr = 0.7$ as described in Section 6.5.



(b) Visualisation of a thermally heated manikin in a cold room.

Figure 5.5.: Visualisation of simulation results on the AESOP system at KAUST (cf. Appendix B.3.1).

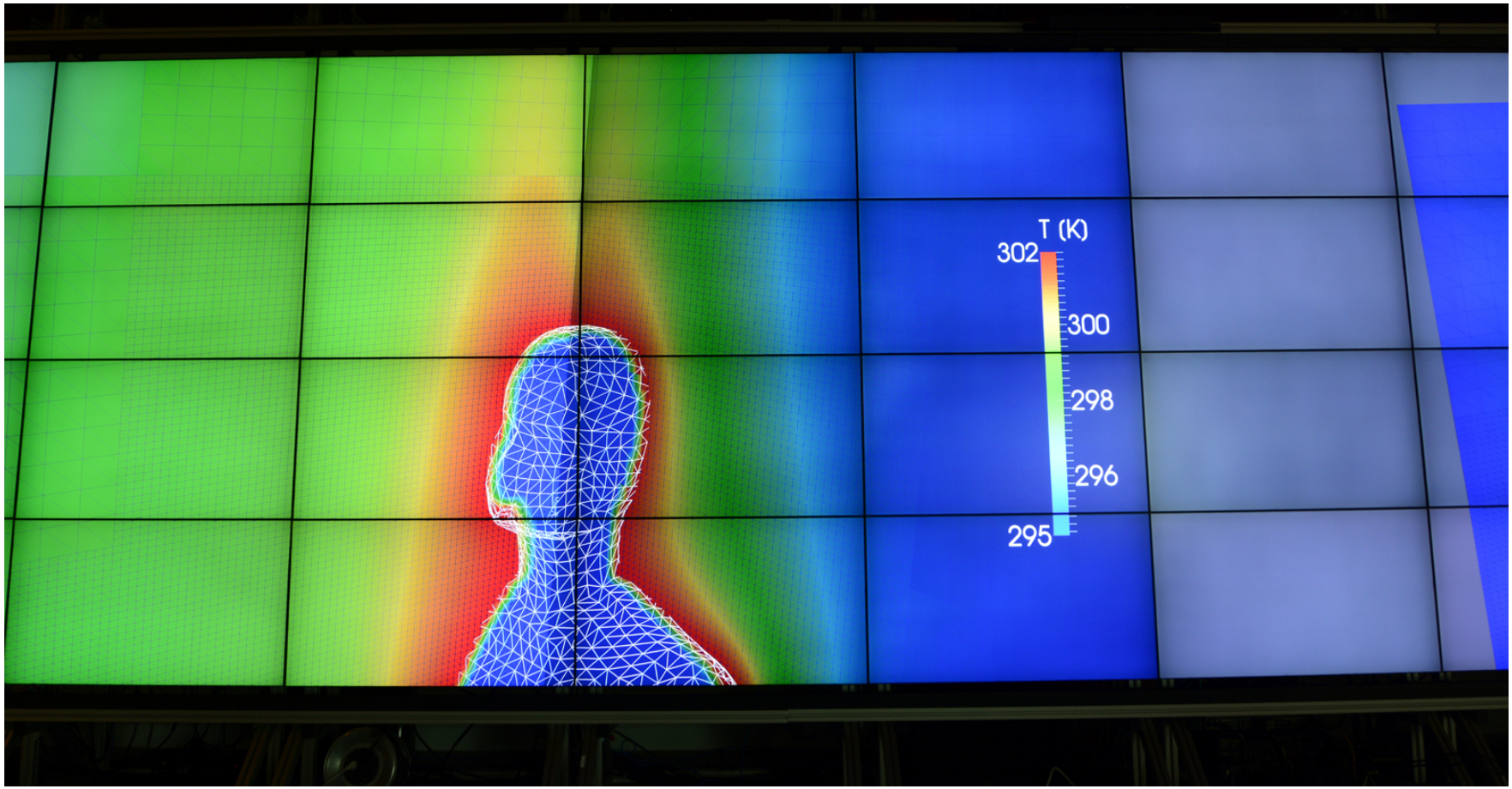


Figure 5.6.: Visualisation of simulation results on the AESOP system at KAUST (cf. Appendix B.3.1). Close up of the thermal manikin in Figure 5.5(b) displayed now in wireframe mode. Effects of l-grid refinements can now be highlighted during this visualisation process.

6. Verification and Validation

In order to determine the correct and expected behaviour of the implemented code, verification and validation tests have to be performed, where the simulated results are compared to results published in literature.

Thus, a few classic, well-studied benchmarks were selected in order to test distinctive features of the code. This chapter introduces the different benchmarks and their peculiarities before presenting the simulated results as well as a discussion and comparison to literature references.

6.1. Lid-Driven Cavity

Ever since the introduction of numerical methods, the lid-driven cavity example served as a model problem for evaluating the accuracy and the behaviour of incompressible fluid flow implementations, even if the problem setup itself has singularities at two of the corners (cf. [GGS82]).

It has no inflow or outflow boundary conditions and the fluid flow is purely shear-driven by the top lid of the cavity. Hence, viscous effects especially can be studied using this problem setup. Furthermore, thermal coupling has been disabled and the flow is regarded as isothermal, i. e. no buoyancy effects are present.

In the following sections, 2D and 3D results of the cavity flow are investigated and the corresponding results are discussed.

6.1.1. 2D Scenario

The 2D setup is depicted in Figure 6.1 and consists of a 1×1 m cavity, where all boundaries are set to *no-slip* boundary conditions as described in Section 2.5. The walls are impenetrable and, additionally, the top wall is given a velocity of 1 m/s in the positive x-direction driving the fluid motion in the domain.

For the discretisation of the computational domain, a uniform refinement with a block size of 32×32 and a recursive bisection was chosen at a depth of 4, resulting in a domain of 512×512 cells, which is regarded as sufficiently fine enough for comparing the simulated results to the references given in Ghia et al. [GGS82]. First results (although not computed

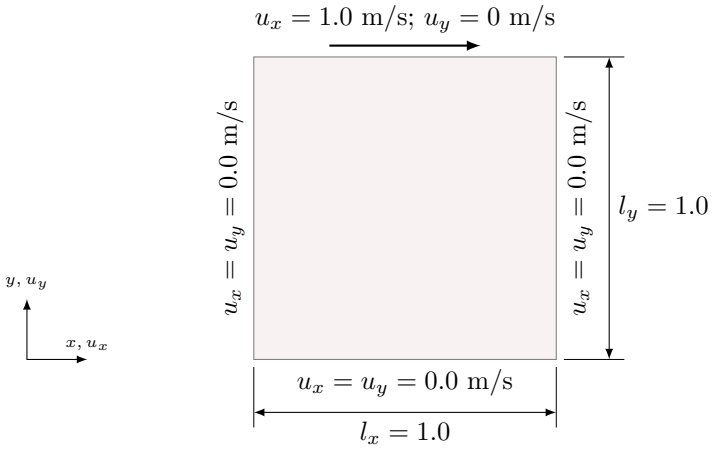


Figure 6.1.: 2D driven cavity setup.

at this resolution) were already presented in [FMR11a]. It should be mentioned at this point that some values in the tables presented in [GGS82] had some obvious typos, which were corrected while comparing the results. One value in the x-axis cutline of the u_y velocity is nevertheless so far off that it is considered to be a mistake, but was not corrected in the diagrams.

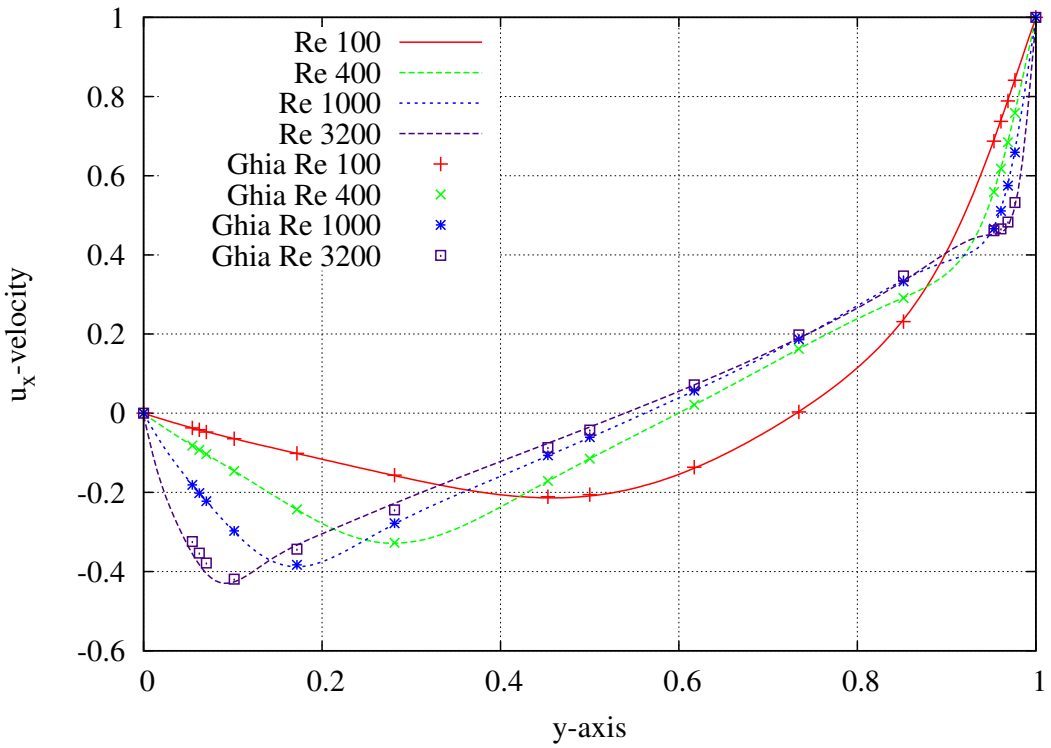
The benchmark solutions are given for different Reynolds numbers, thus describing different types of fluids. The higher the Reynolds number, the less viscous the behaviour of the fluid. As the code does not use dimensionless quantities, the Reynolds number is used for the computation of the kinematic viscosity ν. With the given characteristic length $L_0 = 1$ m and characteristic velocity $v_0 = u_x = 1$ m/s, the viscosity ν can be computed using the modified Equation (2.22) as

$$\nu = \frac{v_0 \cdot L_0}{Re} \quad . \tag{6.1}$$

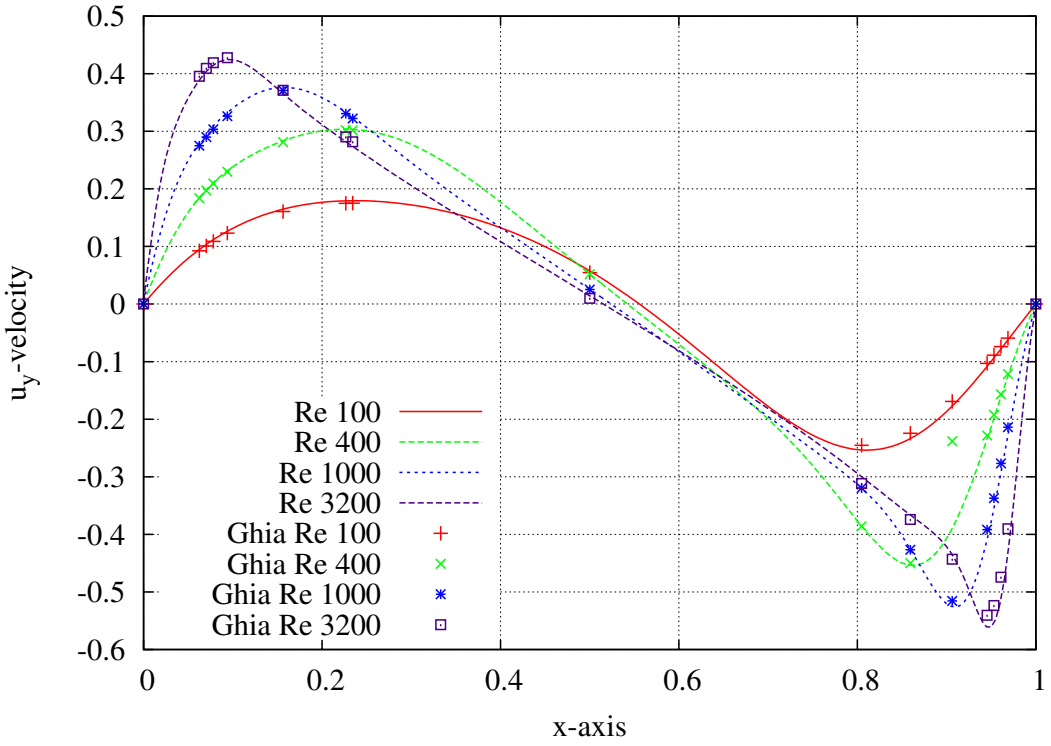
Figure 6.2 shows simulated results as lines, and reference values from tables published in [GGS82] as points, for two different cutlines through the 2D domain at $x = 0.5$ (Figure 6.2(a)) and $y = 0.5$ (Figure 6.2(b)). All the results are in excellent agreement with the reference solutions.

Figure 6.3 depicts plots for different Reynolds numbers containing the velocity magnitude as colour code and using the line integral convolution (LIC) technique [CL93] in order to visualise the flow behaviour. A 'visual comparison' to Figure 3 of [GGS82] shows also excellent agreement of the results.

As a last remark, it should be mentioned that the code actually executed a 3D computation where the resolution in the third dimension was fixed to one cell and the computational kernel was modified using a pre-processor flag in order to ignore the third dimension's boundary conditions. Using this method, the exact same code and data structure can be applied for either 2D or 3D computations. Technically, it could also be used for 1D computations,

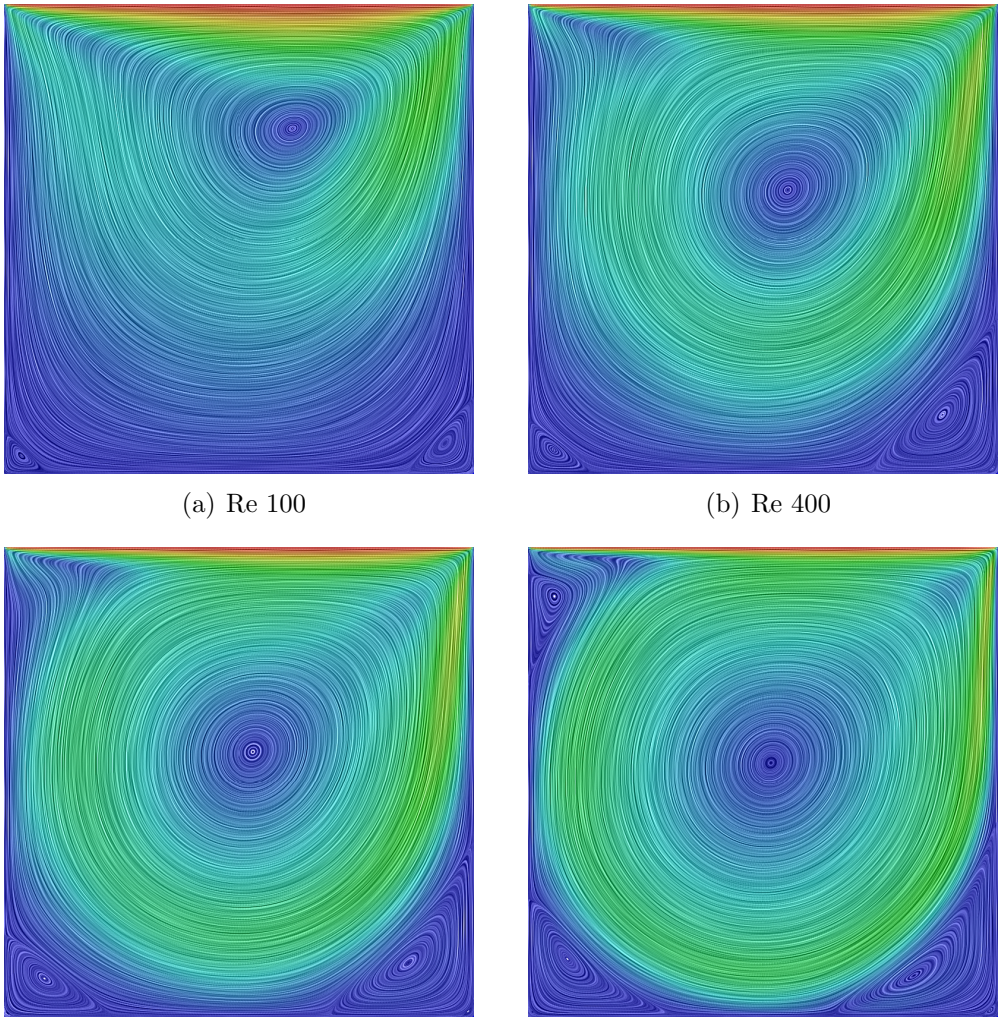


(a) u_x velocity component at $x = 0.5$.



(b) u_y velocity component at $y = 0.5$.

Figure 6.2.: Simulated 2D velocity profiles through the lid-driven cavity example (lines) compared with the results published by Ghia et al. [GGS82] (points).



(a) Re 100 (b) Re 400

(c) Re 1,000 (d) Re 3,200

Figure 6.3.: Simulated lid-driven cavity examples computed in 2D for different Reynolds numbers depicted using line integral convolution (LIC) and overlaid with a velocity magnitude colour code (comparable to Figure 3 from Ghia et al. [GGS82]).

although the data structure presents a large overhead in this case and a specially designed 1D structure would offer more advantages.

6.1.2. 3D Scenario

In order to test the 3D implementation of the code, a $1 \times 1 \times 1$ m cavity is simulated using a 3D setup as depicted in Figure 6.4. All boundary conditions are set to no-slip and the top (x, z)-face at $y = 1.0$ moves with a constant velocity of $u_x = 1.0$ m/s.

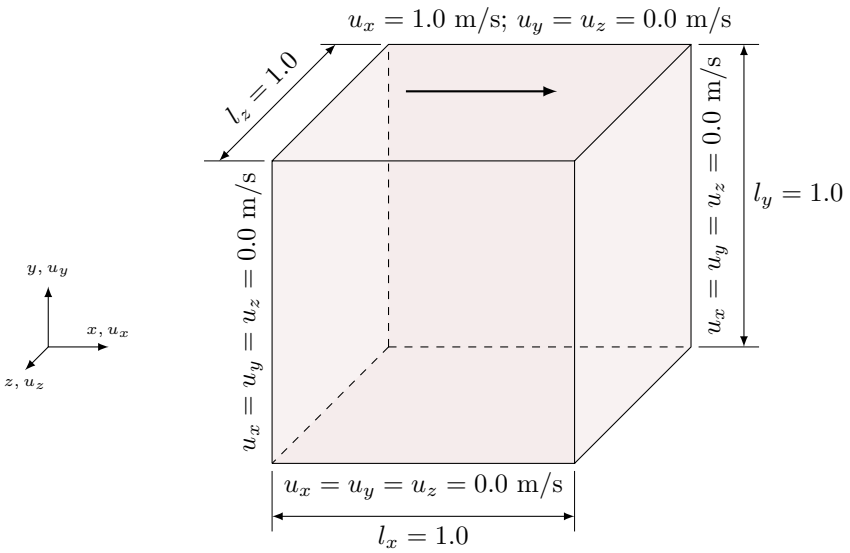


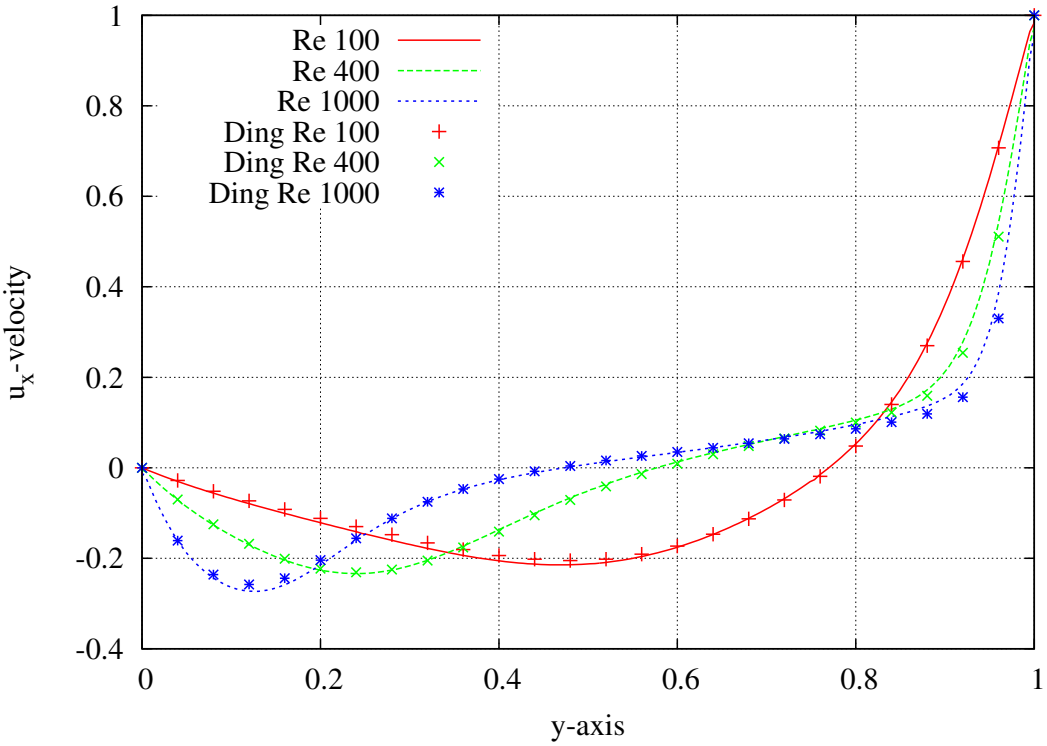
Figure 6.4.: 3D driven cavity setup.

As in the 2D case, the flow is driven purely by viscous forces introduced by the moving lid. Reference solutions to 3D lid-driven cavities can be found in [KHT87] or [DSYX06], for example.

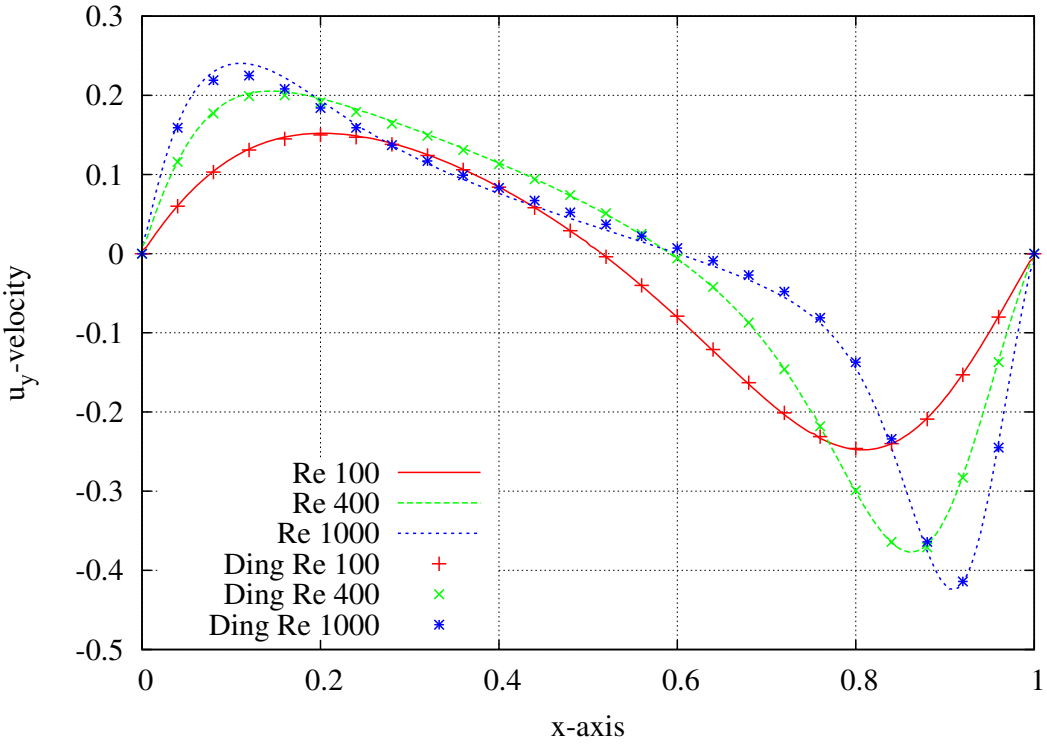
For the simulation results presented, a uniform domain discretisation using a recursive bisection up to depth 2 was chosen with a block size of $28 \times 28 \times 28$ leading to a total of 112^3 (i. e. 1.4 million) cells.

Figure 6.5 shows a comparison of the simulation results with values published by Ding et al. [DSYX06]. As in the 2D case, the results show very good agreement with the references, although the results in Figure 6.5(b) for $Re = 1{,}000$ are a bit off. By looking at Figure 3(c) of the original publication from Ding et al., where they compare their results to those of Ku et al. [KHT87], it can be seen that Ding underestimates the u_y velocities. As the results presented overestimate the u_y velocity by the same factor, it can be assumed that the results are in excellent agreement with the results from Ku et al. As those are unfortunately not presented in tabular form in the paper itself, they could not be used as a reference here.

As a meaningful 3D representation of the 3D lid-driven cavity flow is quite difficult to create, Figure 6.6 shows different slices through the domain for the three previously compared

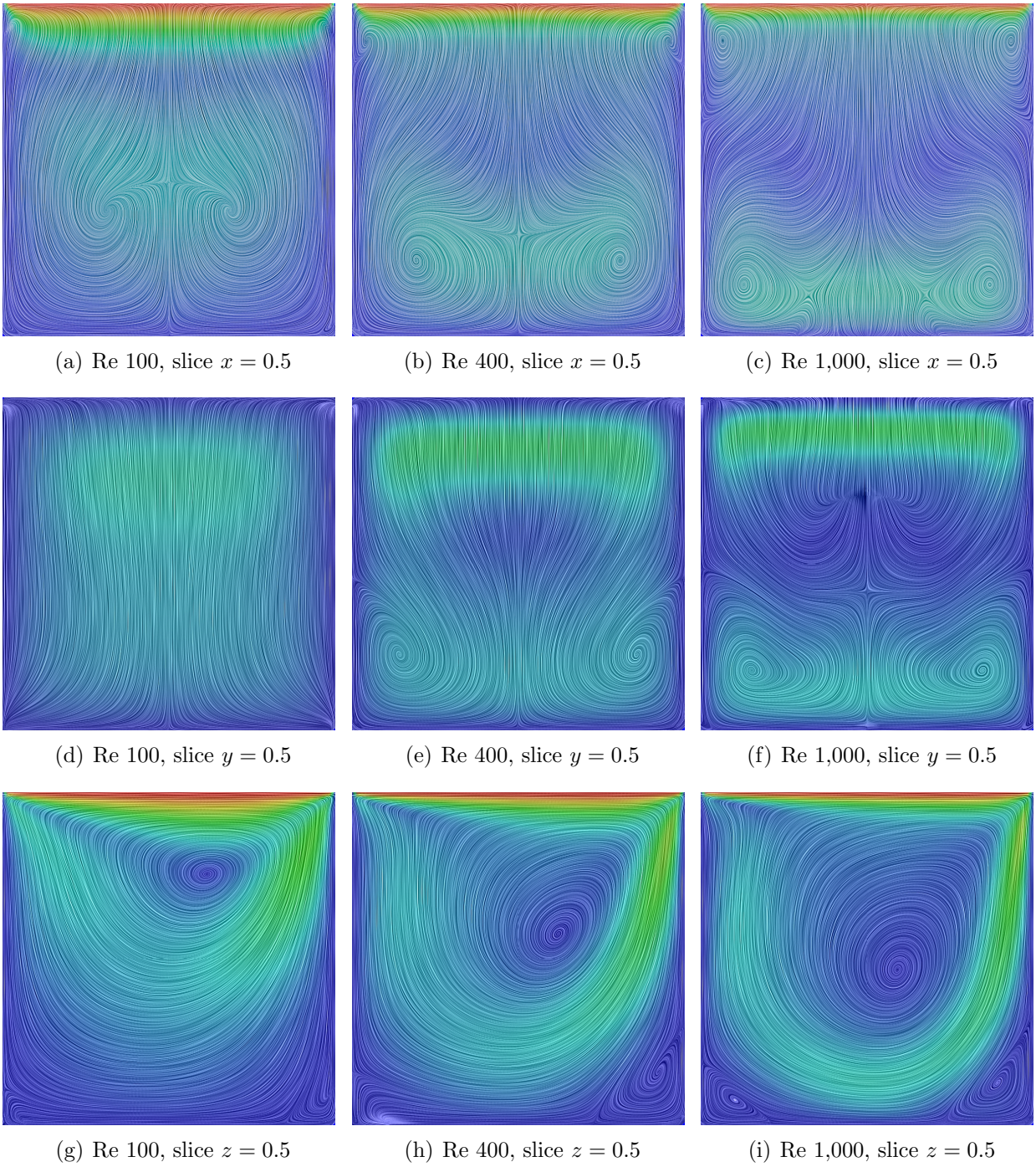


(a) u_x velocity component at $x = 0.5$ and $z = 0.5$.



(b) u_y velocity component at $y = 0.5$ and $z = 0.5$.

Figure 6.5.: Simulated 3D velocity profiles through the lid-driven cavity example (lines) compared with the results published by Ding et al. [DSYX06] (points).



(a) Re 100, slice $x = 0.5$ (b) Re 400, slice $x = 0.5$ (c) Re 1,000, slice $x = 0.5$

(d) Re 100, slice $y = 0.5$ (e) Re 400, slice $y = 0.5$ (f) Re 1,000, slice $y = 0.5$

(g) Re 100, slice $z = 0.5$ (h) Re 400, slice $z = 0.5$ (i) Re 1,000, slice $z = 0.5$

Figure 6.6.: Different slices for simulated lid-driven cavity examples computed in 3D for different Reynolds numbers depicted using LIC and overlaid with a velocity magnitude colour code (comparable to Figures 4, 5, and 6 from Ding et al. [DSYX06]).

Reynolds numbers 100, 400, and 1,000. The top row (i. e. Figures 6.6(a), 6.6(b), and 6.6(c)) presents a cut-plane at $x = 0.5$, the middle row (i. e. Figures 6.6(d), 6.6(e), and 6.6(f)) shows a cut-plane at $y = 0.5$, and the bottom row (i. e. Figures 6.6(g), 6.6(h), and 6.6(i)) represents a cut-plane at $z = 0.5$. The slice through $z = 0.5$ is closest to what could be seen in the 2D case, where similar vortexes could be observed at the bottom of the cavity, although 3D effects influence the positions of the vortexes slightly. The 3D effects can be observed very well in the other two cutting directions. Depending on the Reynolds number, vortexes which influence the behaviour of the main vortex seen in the z-slice start to form at the edges of the cavity.

Performance Measurements

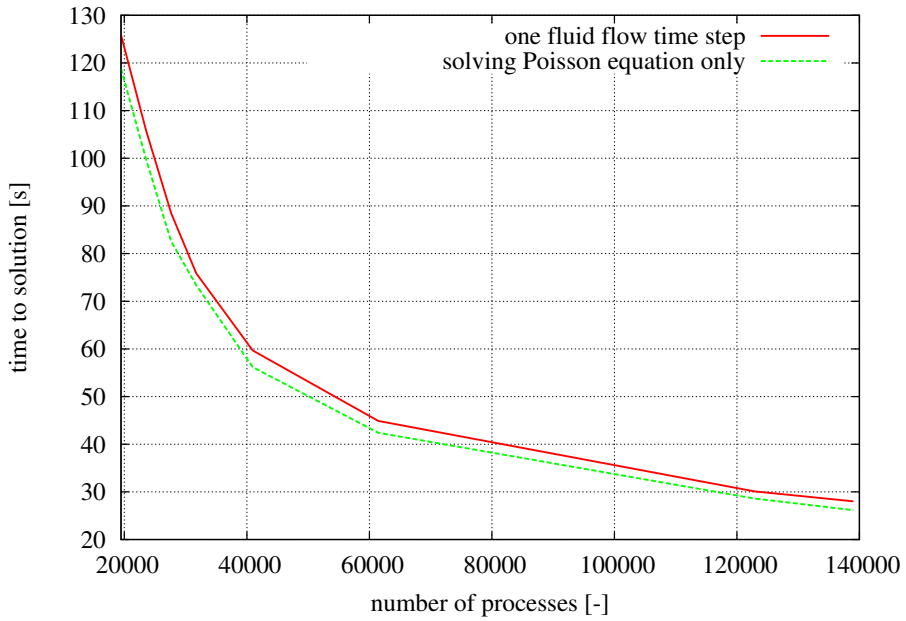


Figure 6.7.: Comparisons of time to solution for the Poisson equation and for solving one fluid flow step of the 3D driven cavity example on SuperMUC at depth 8 using a recursive bisection and a block size of 16×16×16 (thus using 80 billion cells).

In order to estimate the performance of the complete fluid code, measurements for a 3D setup were performed on SuperMUC for up to 139,008 computing processes. In Figure 6.7, a comparison of the time to solution for one time step of the driven cavity setup is shown together with the time to solution for solving the Poisson problem stated in Section 4.6.2.

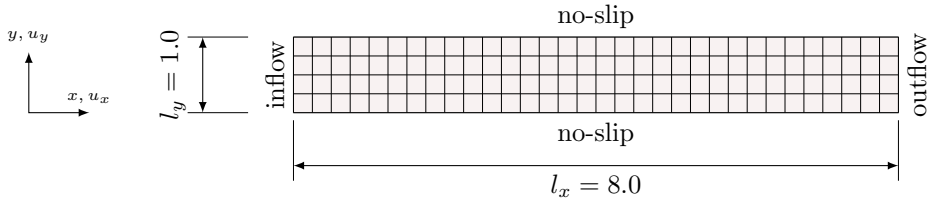
Figure 6.7 indicates that the solution of the Poisson equation takes up around 93% to 96% of the solution time for one time step and the proportion remains more or less constant. Thus, the solution of the Poisson equation presents the main working package of the code presented, and the performance measurements made solely for the pure Poisson equation in Section 4.6 can be regarded as a general trend for the complete code.

6.2. Channel Flow

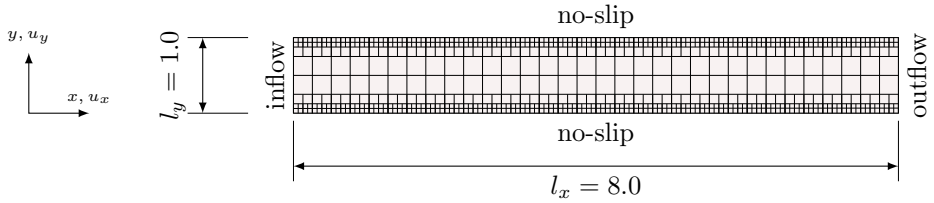
As the viscous fluid behaviour as well as the no-slip boundary conditions provide results as expected on a uniformly refined l-grid, inflow and outflow boundary conditions are added in order to simulate a simple channel flow without any internal obstacles. For this setup, an analytical solution is available and described in detail in [Whi91], for example.

6.2.1. 2D Scenario

The basic setup of a 8×1 m channel is depicted in Figure 6.8. The left-hand side (west) boundary condition at $x = 0$ is defined as inflow and the right-hand side (east) boundary condition at $x = 8$ as outflow. The other boundary conditions are set to no-slip, i. e. impenetrable with vanishing velocities at the boundaries.



(a) uniform setup



(b) adaptive setup

Figure 6.8.: 2D channel setup.

The inflow velocity at $x = 0$ is fixed to $u_{x,in} = 1.0$ m/s and $u_{y,in} = 0.0$ m/s uniformly over the complete side for all $y \in [0, 1]$, whereas the outflow conditions are set in accordance with Section 2.5 as $\partial u_x/\partial x = \partial u_y/\partial y = 0$. Similar to the driven cavity setup, the kinematic viscosity ν is determined for a specific Reynolds number according to Equation (6.1) with $L_0 = l_y = 1.0$ and $v_0 = u_{x,in} = 1.0$ m/s.

Using these boundary conditions and settings, the analytical solution according to [Whi91] is a parabolic profile defined by

$$u_x(\tilde{y}) = \frac{6Q}{l_y^3}\left(\frac{l_y^2}{4} - \tilde{y}^2\right) \quad , \tag{6.2}$$

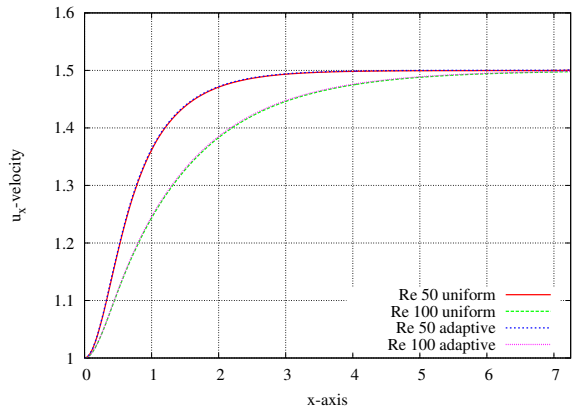
where $Q = l_y \cdot u_{x,in}$ is the volumetric flow rate, assuming a constant inflow profile. Furthermore, this equation assumes the $\tilde{y}$ axis in the range of $\tilde{y} \in [-l_y/2, l_y/2]$, having its maximal value at $\tilde{y} = 0$. In the later diagrams, $\tilde{y}$ is transformed to y in order to be in the same range as the simulated results ($y \in [0, 1]$).

The parabolic flow is not reached immediately, however, but needs a certain distance from the uniform inflow until it is fully developed. An empirical equation in [Whi91] defines the entrance length L_e for a 2D channel as

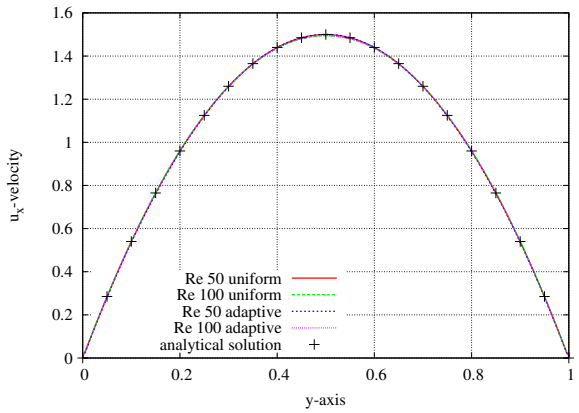
$$L_e \approx l_y \cdot \left(\frac{1}{2} + \frac{Re}{20}\right) \quad . \tag{6.3}$$

Thus, for a channel with Reynolds number 50 or 100, the entrance length is estimated to be 3.0 m or 5.5 m, respectively.

The channel benchmark will be used for several different analyses. First of all, a uniformly refined l-grid is used in the simulation (cf. Figure 6.8(a)). In a second step, an adaptive refinement towards the no-slip boundaries is used (cf. Figure 6.8(b)).



(a) u_x velocity component over a longitudinal cutline through the channel at $y = 0.5$



(b) u_x velocity component over a cross-section cutline through the channel at $x = 6$

Figure 6.9.: Cutlines through a 2D channel of height 1 m and length of approximately 8 m.

The uniform channel was computed using a refinement of (4,1) in the root l-grid and a subsequent refinement of (2,2) for all higher depths up to level 3. The block size was set to (56,28) giving a total of 896 × 112 cells (or 100,352 cells) on the finest level. The adaptive channel with the refinements towards the edges uses the same refinement strategies for the l-grid. It was refined uniformly up to level 3 plus twice a subsequent refinement towards the edges. The block size chosen was (24,12). This d-grid setup has around 101,376 cells and, thus, it is comparable to the uniform setup. If the adaptive d-grid were to be refined on the finest depth 5, it would correspond to 1,536 × 192 cells (or 294,912 cells) and would need roughly three times more elements.

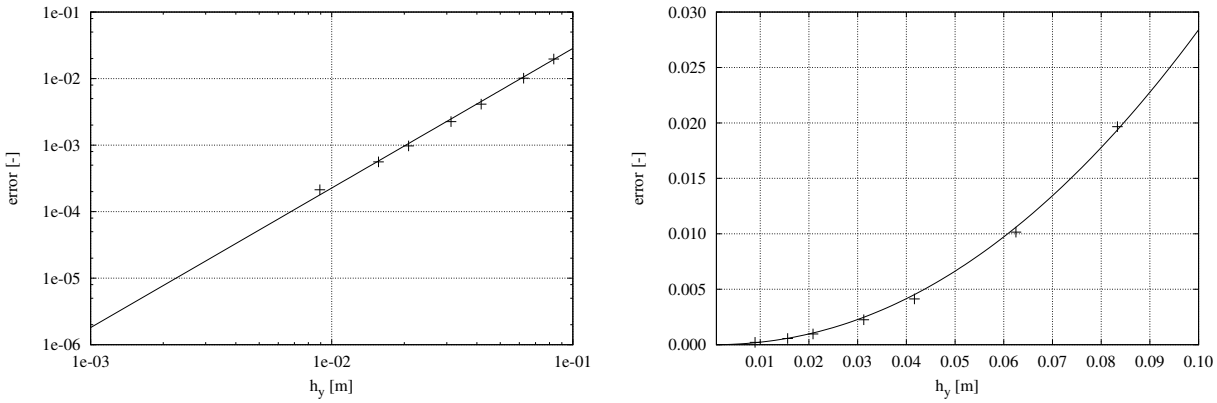
The results for Reynolds number 50 and 100 are depicted in Figure 6.9. In Figure 6.9(a) a longitudinal cutline through the channel can be seen at $y = 0.5$, thus depicting values at

the position where the reference solution reaches its maximal value. First of all, one can see that the empirical equation (6.3) for the entrance length L_e is quite accurate, although for a higher Reynolds number of 100, the flow is not yet fully developed at 5.5 m. It can also be seen that less viscous flows (i. e. higher Reynolds numbers) need longer to reach the parabolic flow behaviour.

A cross-section though the channel was plotted for $x = 6$ in Figure 6.9(b), where both flows are (nearly) fully developed. The results of the analytical solution are plotted as points, whereas the curves for the uniform and adaptive simulation are plotted as lines. The results are again in excellent agreement with the reference solution.

Furthermore, it can be observed that, for both Reynolds numbers, the uniform and the adaptive results are the same. A detailed error analysis of a very similar problem was already performed for the multi-grid-like solver in Section 4.6.3 and will not be repeated at this point for the adaptive versus uniform refinement.

A more interesting error analysis at this point concerns the influence of the cell width (i. e. the d-grid spacing) and the corresponding error convergence towards the analytical solution.



(a) error convergence rate over cell width h_y plotted in double log. scale

(b) error convergence rate over cell width h_y

Figure 6.10.: Error convergence for different cell widths h_y in the y-direction (i. e. different discretisations) for the 2D channel measured at $x = 7, y = 0.5$ (i. e. at the maximal velocity in the profile) and compared to the analytical solution of $u_{max} = 1.5$.

The error convergence plot with respect to the cell width is depicted in Figure 6.10. Different resolutions were computed for a channel with the ratio $N_x : N_y = 8 : 1$ and the cell width in the y-direction $h_y = l_y/N_y$ plotted over the error $e(h_y)$, computed as

$$e(h_y) = \frac{|u_x(h_y) - u_{ref}|}{u_{ref}} \quad , \tag{6.4}$$

where $u_{ref} = 1.5$ m/s denotes the analytical reference solution at the position $x_{ref} = 7$ and $y_{ref} = 0.5$, i. e. the position where the maximal value of the parabolic profile is reached for

a fully developed flow, and $u_x(h_y)$ is the velocity of a simulation with a given cell size h_y at the same position (x_{ref}, y_{ref}).

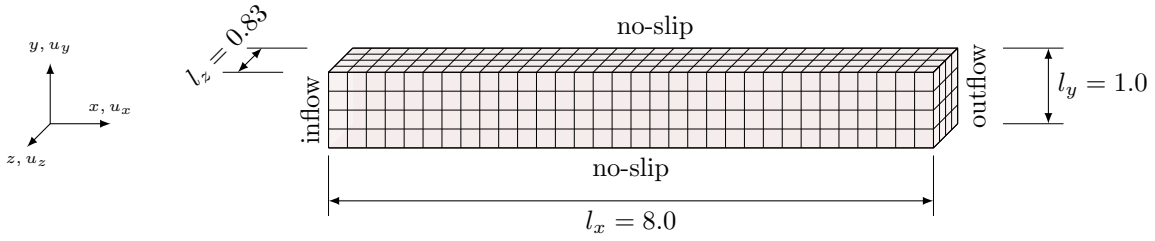
An exponential curve was fitted through the resulting points and yielded the function

$$e(h_y) = 3.5599 \cdot h_y^{2.0981} \quad , \tag{6.5}$$

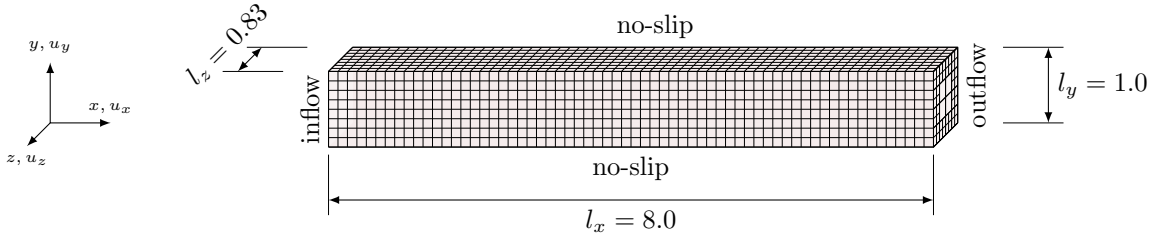
which shows a quadratic error convergence, as expected from applied numerical discretisations and interpolations of second order.

6.2.2. 3D Scenario

Similar to the driven cavity scenario, the channel is computed as a 3D domain in order to test the 3D implementation. In order to check for asymmetries, a different size for the y- and z-direction is chosen in this setup.



(a) uniform setup



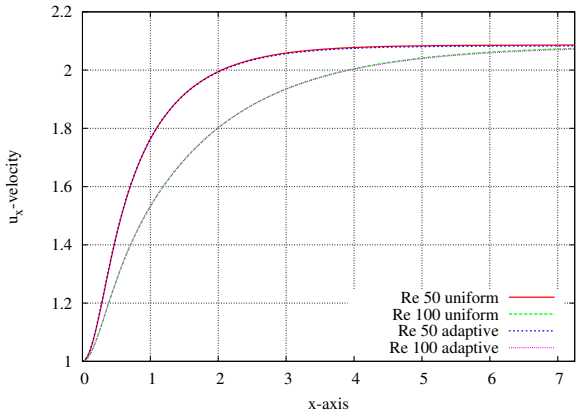
(b) adaptive setup

Figure 6.11.: 3D channel setup.

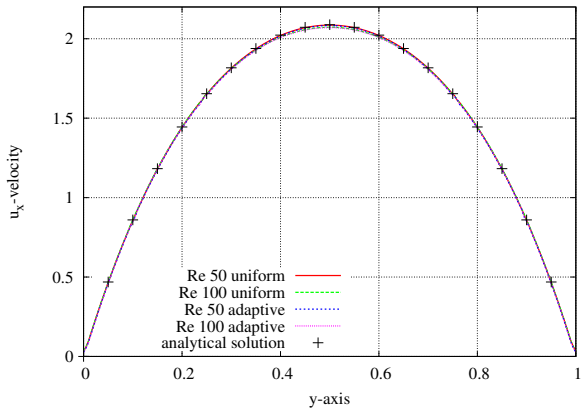
Figure 6.11 shows the general setup of the 3D channel benchmark. West and east boundaries are set as inflow and outflow, respectively, whereas the four remaining boundary conditions are treated as no-slip conditions. The inflow velocity $u_{x,in}(0, y, z) = 1.0$ m/s is assumed constant over the complete inflow face at $x = 0$ for all $y \in [0, 1]$ and $z \in [0, 0.83]$, thus delivering a similar setup as in the 2D case.

The uniform l-grid (cf. Figure 6.11(a)) uses a refinement of (4,1,1) on the root level and a subsequent refinement of (2,2,2) for all higher depths up to level 3 with a block size of (36,18,16) yielding a total domain discretisation of $576 \times 72 \times 64$ (or 2.6 million) cells and a total of 293 l-grids. The adaptive setup (cf. Figure 6.11(b)) has the same l-grid structure

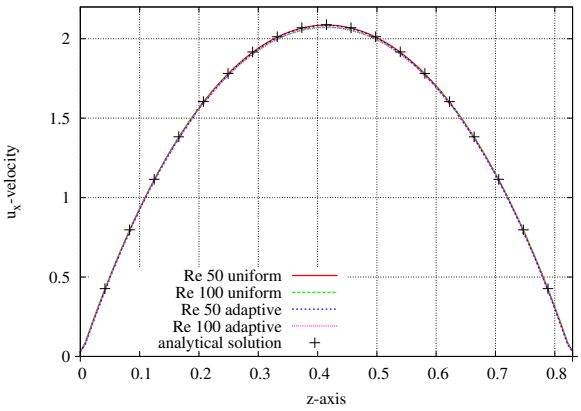
refined up to level 3 with one subsequent refinement towards the no-slip boundaries in the y- and z-direction and uses a block size of (20,10,8) and thus has a total number of 2.56 million cells and 1,829 l-grids. If this setup were to be refined uniformly up to depth 4, it would have a resolution of $640 \times 80 \times 64$ (or 3.3 million) cells.



(a) u_x velocity component over a longitudinal cutline through the channel at $y = 0.5$ and $z = 0.415$



(b) u_x velocity component over a cross-section cut through the channel at $x = 7$ and $z = 0.415$



(c) u_x velocity component over a cross-section cut through the channel at $x = 7$ and $y = 0.5$

Figure 6.12.: Cutlines through a 3D channel of height 1 m, depth of 0.83 m, and length of approximately 8 m.

As in the 2D case, according to [Whi91] an entrance length L_e can be computed until the flow is fully developed by

$$L_e \approx \frac{4A}{P} \cdot \left(\frac{1}{2} + \frac{Re}{20} \right) \quad , \tag{6.6}$$

where $A = l_y \cdot l_z$ is the cross-sectional area and $P = 2l_y + 2l_z$ the wetted perimeter of the channel. Thus, for the given channel setup, the entrance length for Reynolds number 50 and 100 is 2.72 m and 4.99 m, respectively. Figure 6.12(a) shows the velocity component u_x at a longitudinal cut-plane through the mid-axis of the channel at $y = 0.5$ and $z = 0.415$. It can be seen that the estimation of the entrance length is good for $Re = 50$ but far worse for $Re = 100$, where the flow is not yet as far developed. Nevertheless, the estimation is

in the range of a fair approximation. In order to be closer to a fully developed profile, the cross-section of the velocities for comparison is taken at $x = 7$.

The analytical solution for a rectangular cross-section is more complex than in the 2D case and can be written according to [Whi91] as

$$u_x(\tilde{y},\tilde{z}) = \frac{12Q}{\pi^3 ab}\left[1 - \frac{192a}{\pi^5 b}\sum_{i=1,3,5,\ldots}^{\infty}\frac{\tanh(i\pi b/2a)}{i^5}\right]^{-1} \cdot \sum_{i=1,3,5,\ldots}^{\infty}\left[(-1)^{(i-1)/2}\left[1 - \frac{\cosh(i\pi\tilde{z}/2a)}{\cosh(i\pi b/2a)}\right]\frac{\cos(i\pi\tilde{y}/2a)}{i^3}\right] \quad , \tag{6.7}$$

where $Q = 4ab \cdot u_{x,in}$ is the volumetric flow rate, assuming a constant inflow profile. Furthermore, the coordinates are $\tilde{y} \in [-a, a]$ and $\tilde{z} \in [-b, b]$, where $a = 0.5$ and $b = 0.415$ for this specific case. A coordinate transformation from $(\tilde{y}, \tilde{z})$ to (y, z) has to be performed in order to obtain values comparable to the simulated results.

The results of the simulation as well as the analytical solution can be seen in Figures 6.12(b) and 6.12(c) for cross-sections through the y and z axis, respectively. The simulated values for the uniform as well as the adaptive setup are again in excellent agreement with the reference solution.

6.3. Schäfer-Turek Benchmarks – von Kármán Vortex Street

In the DFG priority research program 'Flow Simulation on High-Performance Computers' a collection of benchmarks was generated and presented by Schäfer and Turek in [ST96]. The basic setup is a channel similar to that in the previous case, but with a cylindrical or rectangular obstacle in the inflow region of the channel.

A representation of the 2D setup can be seen in Figure 6.13. In 2D there exist three different benchmarks, namely 2D-1 with Reynolds number 20 which produces a steady, stationary solution, 2D-2 with Reynolds number 100 producing an unsteady solution, and 2D-3 also with Reynolds number 100 but with transient inflow conditions. This work focuses on the 2D-1 and 2D-2 benchmark.

In these benchmarks, the fluid properties for a specific Reynolds number are determined by fixing the kinematic viscosity to $\nu = 10^{-3}$ m^2/s and modifying the inflow velocity for the different benchmark setups. In the 2D-1 and 2D-2 case, the inflow velocity is set to

$$u_x(0, y) = 4 \cdot u_m \cdot y \cdot \frac{l_y - y}{l_y^2}\ [\text{m/s}],\ u_y(0, y) = 0\ \text{m/s} \quad , \tag{6.8}$$

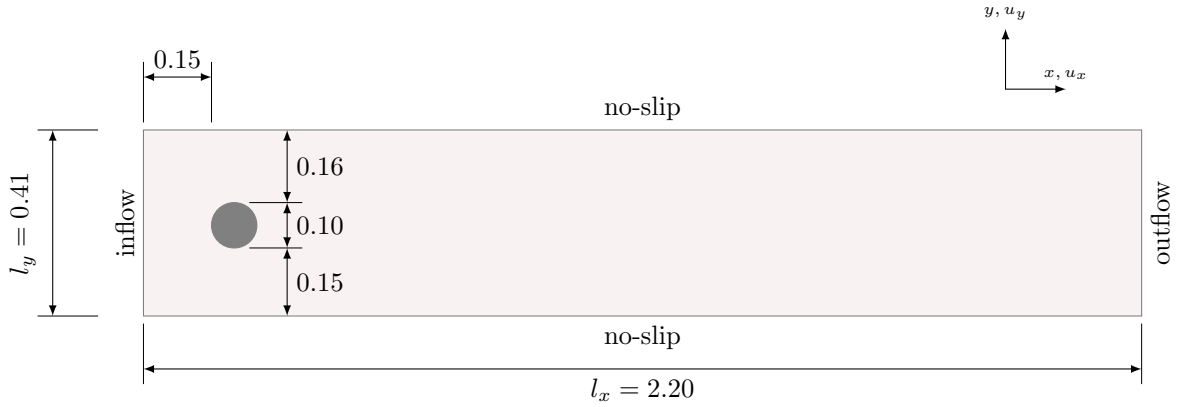


Figure 6.13.: 2D Schäfer-Turek benchmark setup.

with $u_m = 0.3$ m/s producing $Re = 20$ for 2D-1, and $u_m = 1.5$ m/s yielding $Re = 100$ for 2D-2 for $y \in [0, l_y]$. For further details, the interested reader should consult [ST96].

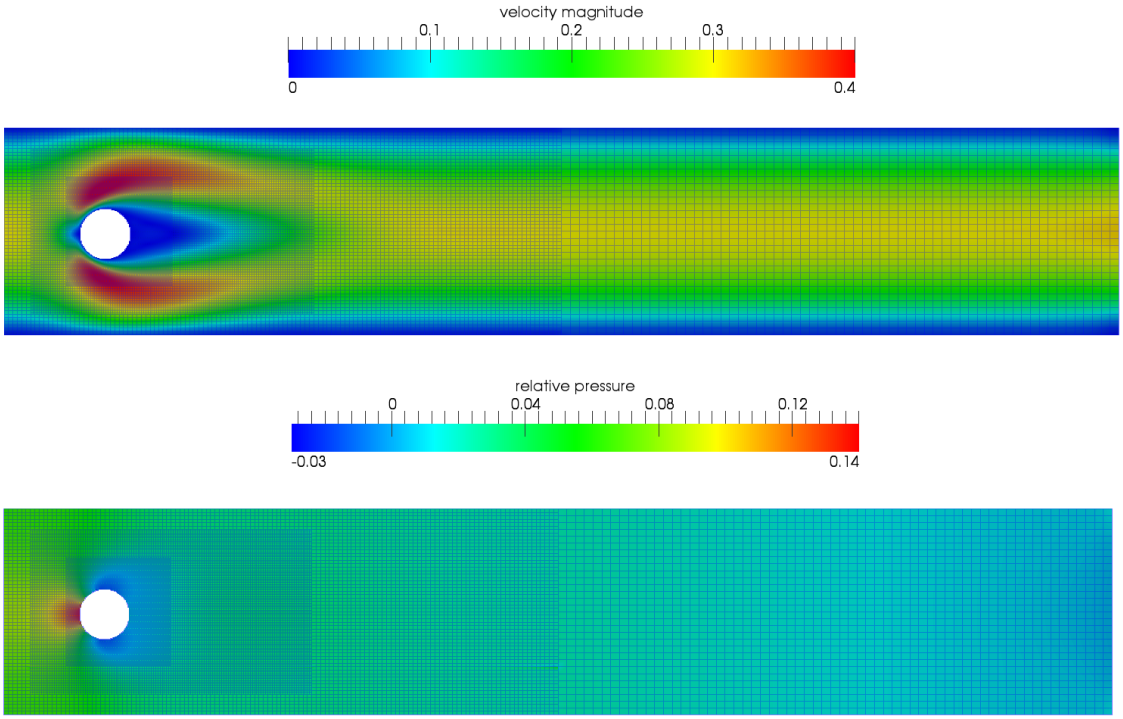


Figure 6.14.: 2D-1 benchmark results for Re = 20 computed on an adaptive l-grid using four different levels (top: velocity magnitude, bottom: pressure distribution).

Figure 6.14 shows the results of a simulation of the 2D-1 benchmark case with an adaptive grid setup around the obstacle region. The l-grid refinement was chosen to be (4,1) on the root level and (2,2) for all higher depths. A uniform refinement was performed until depth 3 with three further adaptive refinements around the obstacle regions. The block size chosen was (16,16), yielding a total number of 28,672 cells. For the pressure difference just in front of and after the cylindrical obstacle, $\Delta p = 0.1162$ Pa was measured. The reference

solution indicates a region of $[0.1172, 0.1176]$. If the pressure difference of the simulation is compared to the average of the indicated range, an error of around 1% was observed, which is acceptable. The recirculation length was measured to be $L_a = 0.0821$ m and should be in the range $[0.0842, 0.0852]$ and, thus, has an average error of 3.1% from the benchmark values. It should be noted here that 15 different codes with different resolutions and solving techniques had results in a much broader spectrum, and that the simulation results presented are still well within the range of those of the other programs (cf. Table 3 in [ST96] for a full list of results).

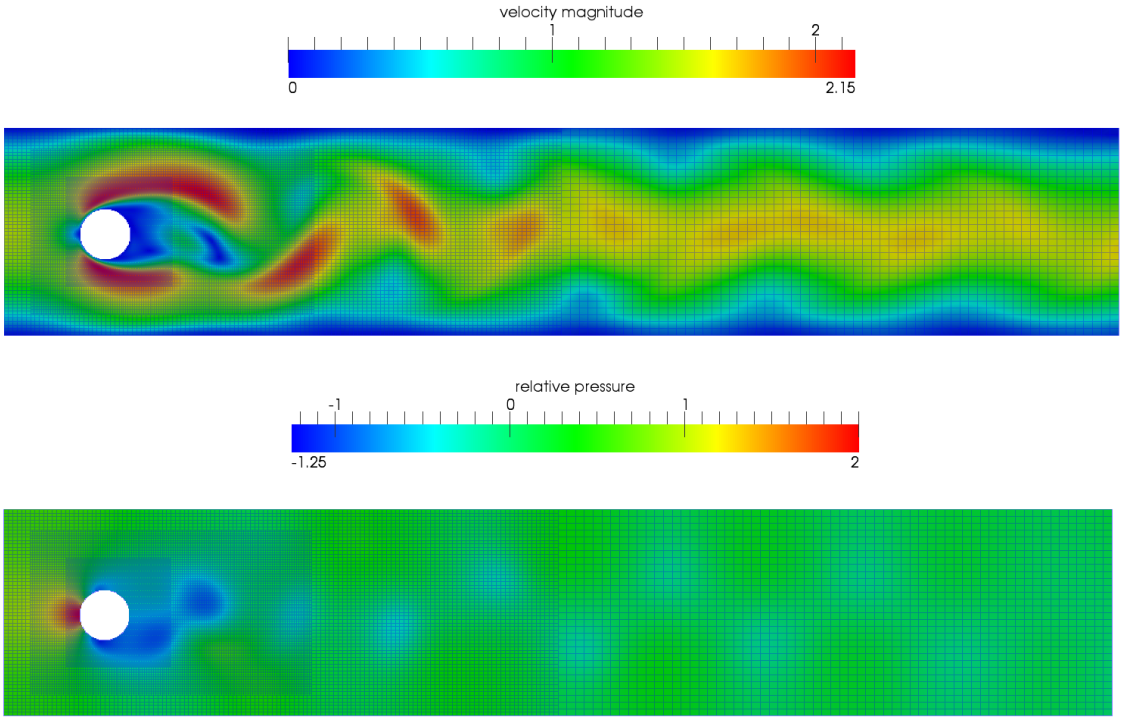


Figure 6.15.: 2D-2 benchmark results for Re = 100 computed on an adaptive l-grid using four different levels (top: velocity magnitude, bottom: pressure distribution).

Figure 6.15 shows the results of the unsteady 2D-2 benchmark. Here, the typical picture of shedding vortexes can be seen, as is known from satellite pictures of the Beerenberg volcano in the north of Jan Mayen[1] Island, for example. The benchmark uses the same domain properties as the 2D-1 benchmark, with the difference of a higher inflow velocity as described in the section above.

As an unsteady state is present, a different kind of evaluation is performed for this scenario. The pressure difference Δp should be computed for one period f of vortex shedding.

Figure 6.16 shows the pressure difference with the evolution over simulation time. Depending on the start point of the measurement, Δp lies in the range $[2.417, 2.501]$. The benchmark indicates a region from $[2.460, 2.500]$ which corresponds excellently to the simulated results.

[1] `http://www.jan-mayen.no`

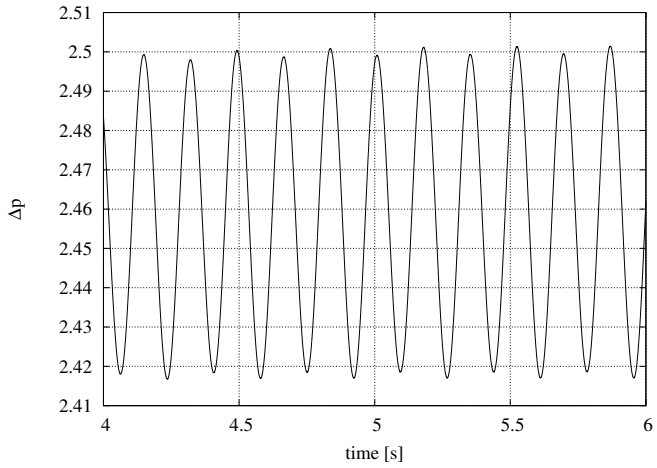
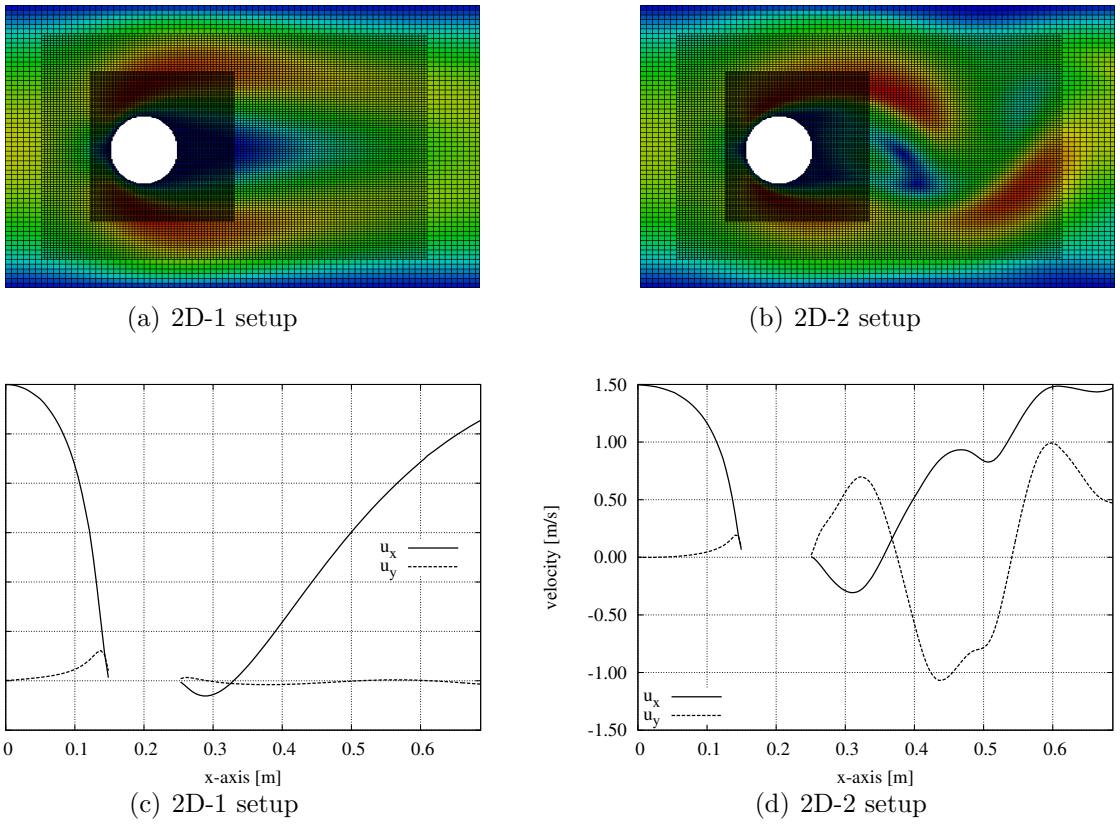


Figure 6.16.: Pressure difference before and after the cylindrical obstacle plotted over the simulation time for the 2D-2 benchmark case.



(a) 2D-1 setup (b) 2D-2 setup

(c) 2D-1 setup (d) 2D-2 setup

Figure 6.17.: Zoom around the refined regions of the velocity plots of Figure 6.14 and 6.15. Lower plots show the velocity components u_x and u_y along a longitudinal cut axis through the centre.

It can be shown that the frequency of the pressure oscillation seen in Figure 6.16 is twice the frequency of the shedding of vortexes f. Hence, if two periods are measured (from $t_b = 4.49247$ to $t_e = 4.83602$, for example), a frequency of $f = 1/(t_e - t_b) = 2.91078$ Hz can be determined. With $\bar{U} = 1.0$ m/s for the 2D-2 benchmark, the Strouhal number St – named after the Czech physicist Vincenc Strouhal – which describes oscillations in fluid mechanics, can be computed as

$$St = \frac{D \cdot f}{\bar{U}} \quad , \tag{6.9}$$

where $D = 0.1$ m is the diameter of the cylindrical obstacle. The Strouhal number is determined here as 0.2911 and should be in the region of $[0.2950, 0.3050]$. Again the average error is in the range of 3%, and again, the results from 10 different codes provide a much broader range around the values computed with the present code.

Figure 6.17 shows excerpts of the velocity plots around the refined regions for the 2D-1 and 2D-2 setup. The velocity components u_x and u_y are plotted against a longitudinal cut along the x-axis through the middle of the domain at $l_y = 0.205$ m and are depicted in Figures 6.17(c) and 6.17(d) for 2D-1 and 2D-2 setup, respectively. It can be observed that the velocity components are smooth in the transition regions from fine regions to coarse regions.

6.4. Convection due to Heated Side-Walls

The previously introduced benchmarks focused on the pure fluid flow validation, as well as inflow, outflow, or no-slip boundary conditions with and without obstacles in the domain. The following benchmarks will investigate effects from the thermal coupling of the advection-diffusion equation (2.29).

The first test case analysed consists of a 2D square cavity with a natural (or free) convection flow induced by hot and cold walls.

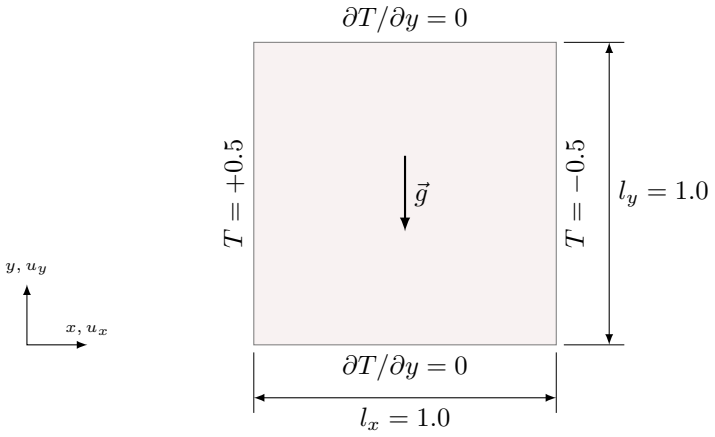


Figure 6.18.: Heated sidewalls benchmark setup.

Figure 6.18 shows the basic setup of the natural convection benchmark. The vertical west wall at $x = 0$ is heated with a fixed normalised (dimensionless) temperature $T = +0.5$, the vertical east wall at $x = 1$ is set to a cold temperature of $T = -0.5$. The horizontal south and north walls at $y = 0$ and $y = 1$, respectively, are perfectly insulated and modelled by homogeneous Neumann boundary conditions $\partial T/\partial y = 0$. Furthermore, all walls are set to no-slip boundary conditions for velocities and pressure settings. The gravitation vector $\vec{g}$ points in the negative y-direction and is set to $\vec{g} = (0, -1)^T$.

As introduced in Section 2.4.2, the Boussinesq approximation is used for coupling the thermal advection-diffusion equation to the momentum equations and certain parameters have to be fixed in order to simulate specific fluid characteristics. The properties of the fluid for a free convective flow can be described by two parameters, namely the Prandtl number Pr introduced in Equation (2.35) and the Rayleigh number Ra introduced in Equation (2.37). This benchmark uses a constant Prandtl number $Pr = 0.71$ and varies the Rayleigh numbers in order to simulate different viscosities and behaviours of the fluid.

As the flow is driven by temperature differences, it is possible to apply 'negative' temperatures for the cold wall. Thus, by setting $T_\infty = 0$, $\Delta T = T_{hot} - T_{cold} = 1$, and $\rho_\infty = 1$, the equations simplify for this benchmark. Furthermore, the as yet undefined thermal expansion coefficient β is fixed to 1. Tests showed that changing the constant $\rho_\infty \cdot \beta \cdot g_i$ in the Boussinesq coupling only affects how fast a quasi-stationary state is reached and not the values themselves to which it converges. As $\rho_\infty \cdot \beta \cdot g_i = 1$ produced stable results in a short simulation time, these values were applied in this setup.

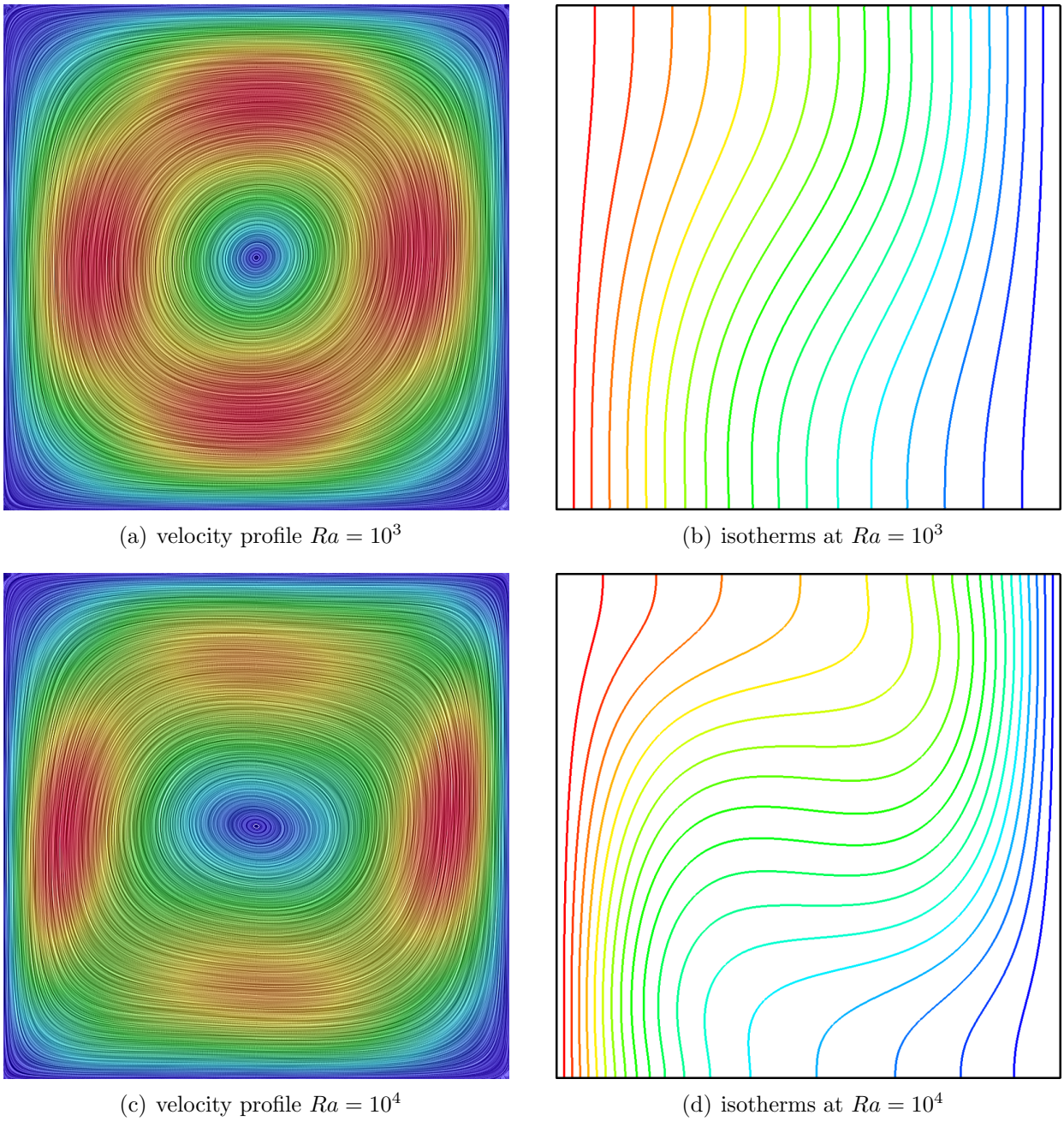
Different resolutions and depths were used as numerical discretisations. If not otherwise specified, a recursive bisection was used over all levels up to depth 3 with a block size of (32,32), yielding a computational domain of 256×256 cells.

Figures 6.19 and 6.20 show simulation results for different Rayleigh numbers. On the left-hand-side, the velocity profile can be seen as colour-coded magnitude overlaid by a LIC structure of the flow profile. On the right-hand-side, isotherms are depicted colour coded from red (hot) to blue (cold) in order to indicate the temperature distribution in the cavity. First comparisons to results published by [DVD83], [Phi84], [ES95], or [vT04] are in excellent agreement with the simulated results.

In order to obtain a better quantification, the so-called *Nusselt number* Nu for the hot wall at $x = 0$ can be calculated for this benchmark according to [ES95] by

$$Nu := -\frac{1}{\Delta T} \int_0^{l_y} \left. \frac{\partial T(x,y)}{\partial x} \right|_{0,y} \mathrm{d}y \quad . \tag{6.10}$$

The Nusselt number characterises the ratio of convective heat transfer to conductive heat transfer. In order to discretise the gradient of the temperature at the warm wall, a higher



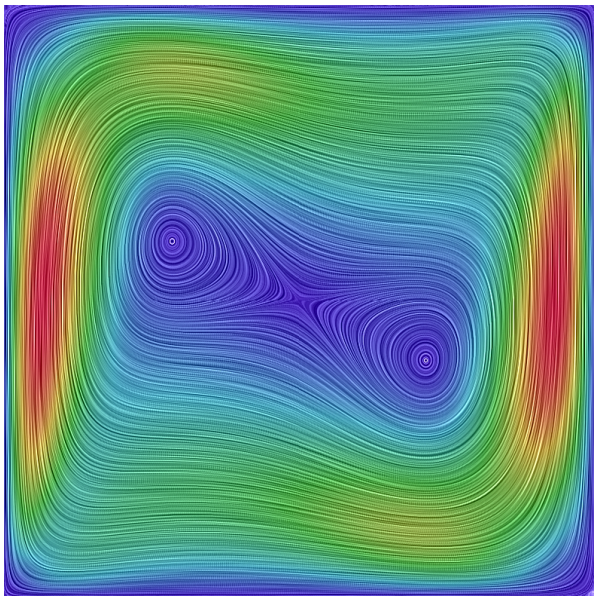
(a) velocity profile $Ra = 10^3$

(b) isotherms at $Ra = 10^3$

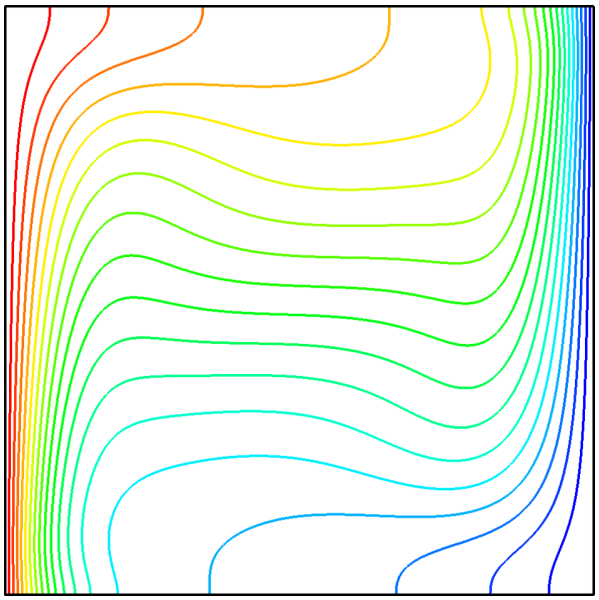
(c) velocity profile $Ra = 10^4$

(d) isotherms at $Ra = 10^4$

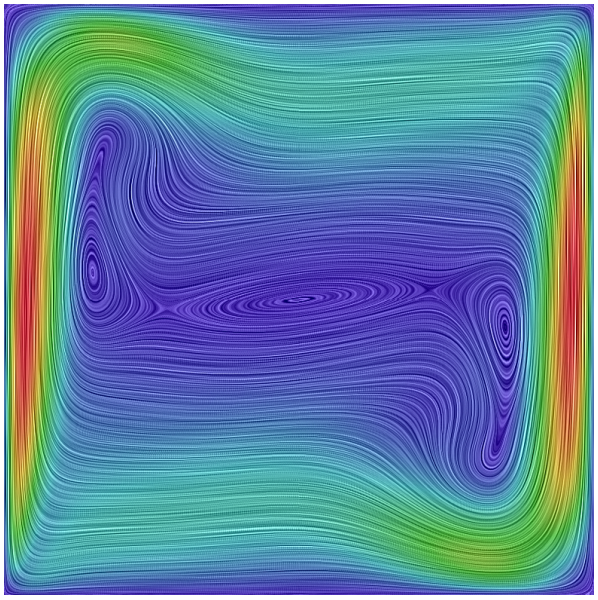
Figure 6.19.: Velocity and isothermal profiles for a square cavity with heated sidewalls (left-hand side warm, right-hand side cold) at $Pr = 0.71$ for different Rayleigh numbers computed on a 256^2 domain resolution.



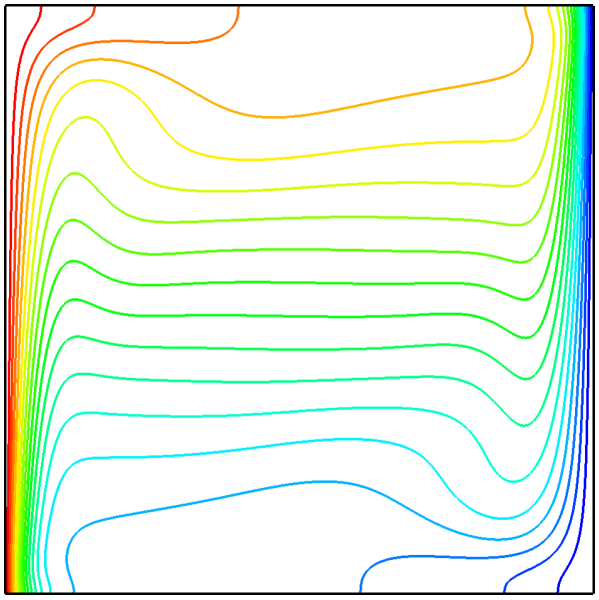
(a) velocity profile $Ra = 10^5$



(b) isotherms at $Ra = 10^5$



(c) velocity profile $Ra = 10^6$



(d) isotherms at $Ra = 10^6$

Figure 6.20.: Velocity and isothermal profiles for a square cavity with heated sidewalls (left-hand side warm, right-hand side cold) at $Pr = 0.71$ for different Rayleigh numbers computed on a 256^2 domain resolution.

order approximation is used following Phillips [Phi84], which states that

$$\left.\frac{\partial T(x,y)}{\partial x}\right|_{0,y} = \frac{-3T(0,y) + 4T(\Delta x, y) - T(2\Delta x, y)}{2\Delta x} \quad , \tag{6.11}$$

where Δx is the cell width used for the current discretisation.

Table 6.1.: Comparison of simulated Nusselt numbers at the hot wall ($x = 0$) to literature results.

Ra	Nu_{sim} (N=128)	Nu_{sim} (N=256)	Ref. [DVD83]	Ref. [LQ91]	Ref. [vT04]
10^3	1.11768	1.11772	1.117	–	1.1170
10^4	2.24561	2.24466	2.238	–	2.2435
10^5	4.53123	4.51976	4.509	–	4.5168
10^6	8.85566	8.78860	8.817	8.8250	8.8239
10^7	15.6709	15.9137	–	16.5230	16.5571

Simulated results of the Nusselt number Nu_{sim} at the hot wall as well as comparative reference values from literature are summarised in Table 6.1. If a simulation was not performed in the corresponding publication, no value is indicated in the table. The simulated Nusselt numbers are given for two different domain discretisations, namely 128^2 and 256^2.

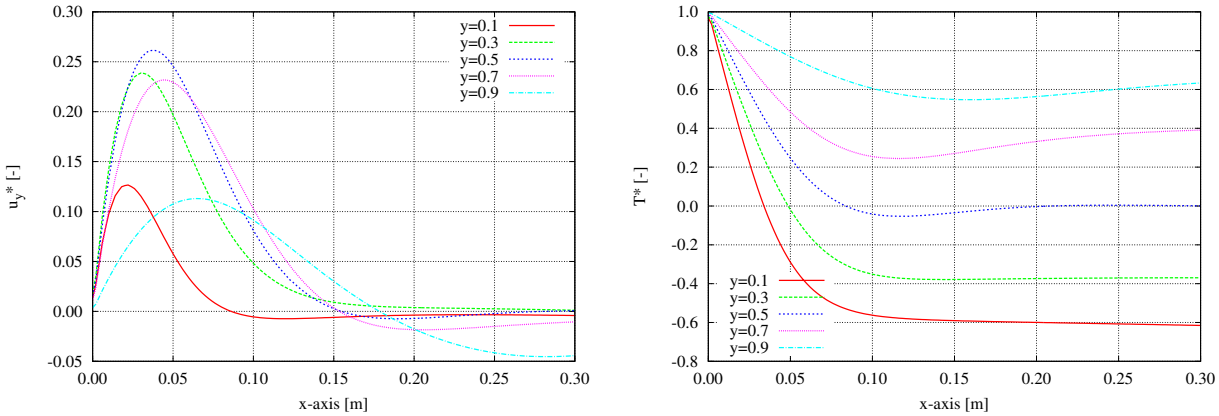
When comparing the simulated results to the values published by [DVD83], the errors for the domain discretisation of 128^2 lie in the range of 0.06% to 0.43% for $Ra = 10^3$ to $Ra = 10^6$, respectively. For 256^2 they range between 0.06% and 0.32%, which shows a great agreement with reference solutions.

For higher Rayleigh numbers of 10^7, however, an error of around 5.2% for a discretisation of 128^2 and 3.7% for a discretisation of 256^2 is relatively high. This is probably due to the fact that the current simulation kernel uses only a standard central difference approximation for the convective terms in the momentum equations. As [GDN98] shows, the convective terms become dominant at low viscosities (i. e. high Reynolds or high Rayleigh numbers) due to the mathematical relation

$$\frac{\partial u^2}{\partial x} = 2u\frac{\partial u}{\partial x} \quad , \tag{6.12}$$

where u now enters as a factor in front of the derivative itself. Thus, for lower viscosities (as in the case of $Ra = 10^7$), a *donor-cell scheme* should be applied, mixing central differences with upwind differences for the convective terms. Details can be found in [HNR75], for example. This is not integrated into the current implementation at the moment and is to be added at some future point.

In order to analyse the results for $Ra = 10^6$ in more detail, a plot containing normalised u_y^* velocities for different horizontal cutlines is shown in Figure 6.21(a). Figure 6.21(b) shows the normalised temperature distribution T^* at the same horizontal cutlines.



(a) normalised velocity (b) normalised temperature

Figure 6.21.: Normalised velocity u_y^* and temperature T^* for $Ra = 10^6$ plotted over different horizontal cutlines through the heated side-walls benchmark for a domain discretisation of 256^2.

The normalised velocity u_y^* in the y-direction is defined according to [ES95] as

$$u_y^* = u_y \cdot (\|\vec{g}\| \cdot \beta \cdot \Delta T \cdot l_y)^{-1/2} \quad , \tag{6.13}$$

and the normalised temperature T^* as

$$T^* = \frac{2 \cdot (T - T_\infty)}{\Delta T} \quad , \tag{6.14}$$

in order to obtain the same values as in the results published by [ES95] or [Jan94].

Comparing Figures 6.21(a) and 6.21(b) to Figure 5 and 6 published in [ES95], excellent agreement can be seen. The maximal normalised velocity u_y^* for the cut through the middle of the cavity at $y = 0.5$ for a discretisation of 240^2 is given by [Jan94] as $u_y^* = 0.2618$, whereas the simulated results yielded $u_y^* = 0.2616$ for a comparable discretisation of 256^2, which represents an error of 0.07% for $Ra = 10^6$ and is outstanding.

Furthermore, the influence of the cell width h_y, i. e. the discretisation of the domain, is analysed with respect to the maximal normalised velocity u_y^*. Figure 6.22 shows the results for different cell sizes as well as literature reference values published by [ES95] and [Jan94]. It can be seen that the values converge towards the same solution if h_y is reduced. It can also be seen that the thermal coupling has a (more or less) quadratic convergence rate, as expected from a system discretised using second order-based methods. Furthermore, it can be seen that the reference values from literature diverge for different solution approaches and the code presented is well inside the range of the published results.

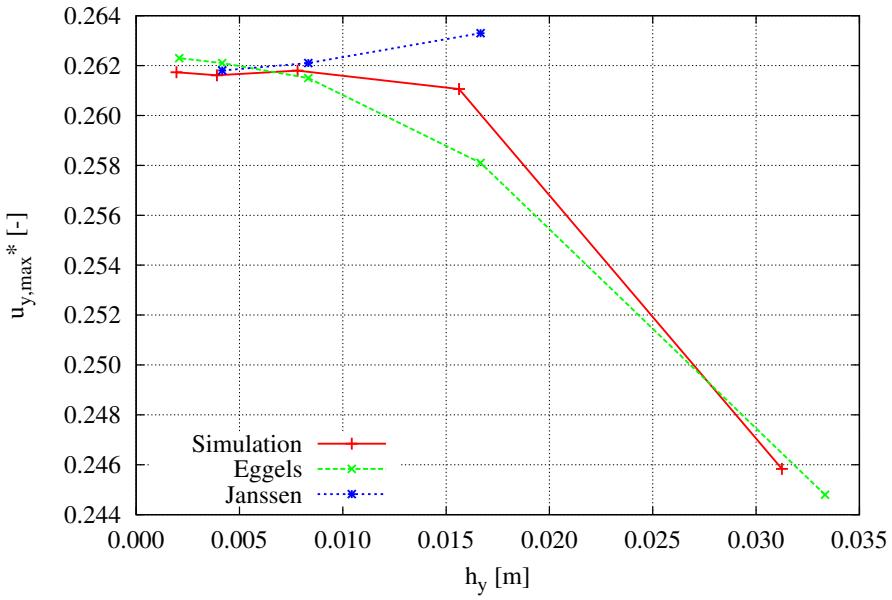


Figure 6.22.: Normalised maximum velocity v^*_{max} for $Ra = 10^6$ in the y-direction measured at a horizontal cutline at $y = 0.5$ for the simulated results and two comparisons from literature.

6.5. Rayleigh-Bénard Convection

A second benchmark for the thermally coupled code has a very similar setup but provides very different results. Basically, the gravitation vector is rotated, so that the bottom wall at $y = 0$ is heated and the top wall at $y = 1$ is cooled. Thus, the temperature gradient is acting against the gravitational force and the resulting flow pattern is called Rayleigh-Bénard convection, analysed by Bénard in [Bén01] and by Lord John William Strutt, 3rd Baron Rayleigh.

The fundamental difference between the previous benchmark using heated side-walls and the Rayleigh-Bénard convection is that a critical Rayleigh threshold has to be reached before the fluid is set in motion by convection phenomena. If the domain is sufficiently long and wide, the critical Rayleigh number is

$$Ra_{crit} \approx 1{,}707.8 \tag{6.15}$$

according to [PS40], who did a detailed analytical analysis of the problem in the 1940s.

A 2D benchmark setup is depicted in Figure 6.23. In order to make the domain 'sufficiently long enough', periodic boundary conditions are chosen at $x = 0$ and $x = 2$.

Periodic boundary conditions are realised in the implementation by manipulating the neighbourhood search described in Section 4.4.1. If a specific direction is modelled as 'periodic', the neighbourhood server delivers the corresponding l-grid on the opposite side of the domain instead of returning an 'out-of-domain' flag. Thus, periodic conditions can be implemented

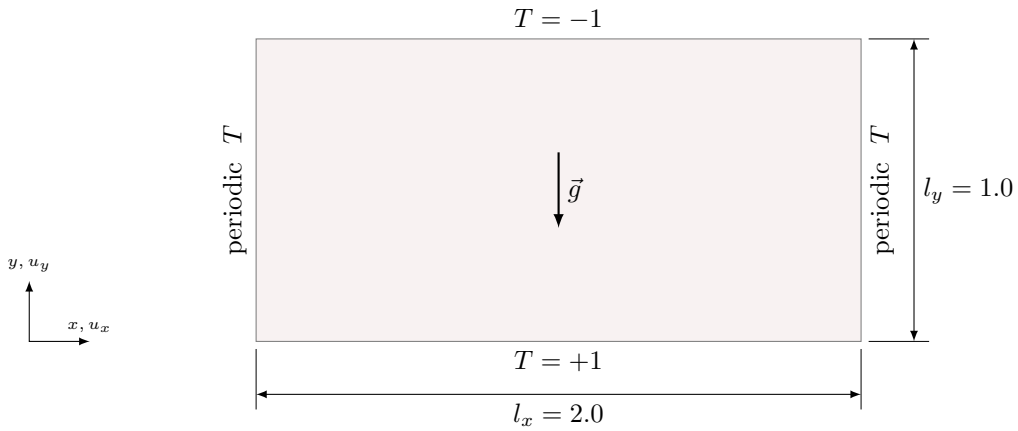
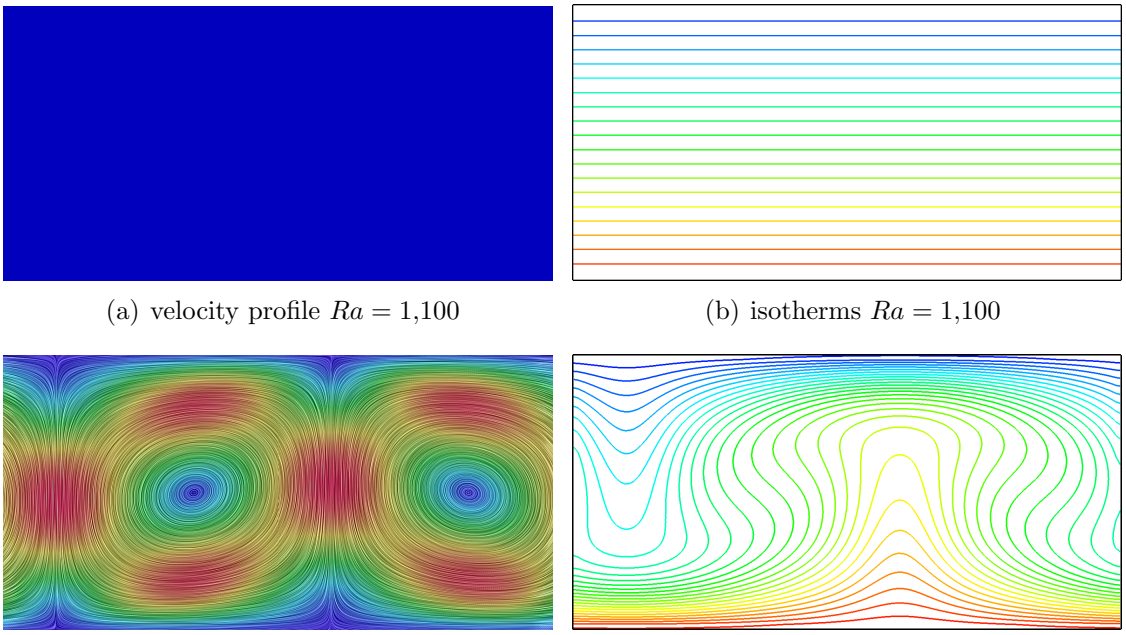


Figure 6.23.: 2D Rayleigh-Bénard benchmark setup.

with approximately three lines of code, and all the copying procedures for the ghost layer halos are performed automatically by the communication phases themselves. The other fluid properties are set as in the previous benchmark except $\Delta T = T_{hot} - T_{cold} = 2$, which is used for this case.



(a) velocity profile $Ra = 1{,}100$

(b) isotherms $Ra = 1{,}100$

(c) velocity profile $Ra = 8{,}000$

(d) isotherms $Ra = 8{,}000$

Figure 6.24.: Velocity and isothermal profiles for a 2:1 cavity with heated floor and cooled ceiling (Rayleigh-Bénard convection setup) at $Pr = 0.71$ for $Ra = 1{,}100$ and $Ra = 8{,}000$ computed on a 64×32 domain resolution. NB: the velocity and temperature colour profile is chosen to be exactly the same for both settings, ranging from $|\vec{u}| \in [0, 0.5]$ m/s and $T \in [-1, 1]$.

Figure 6.24 shows velocity profiles and isotherms for sub-critical $Ra = 1{,}100$ and super-critical $Ra = 8{,}000$. For sub-critical Rayleigh numbers, the fluid is quiescent (i. e. no fluid

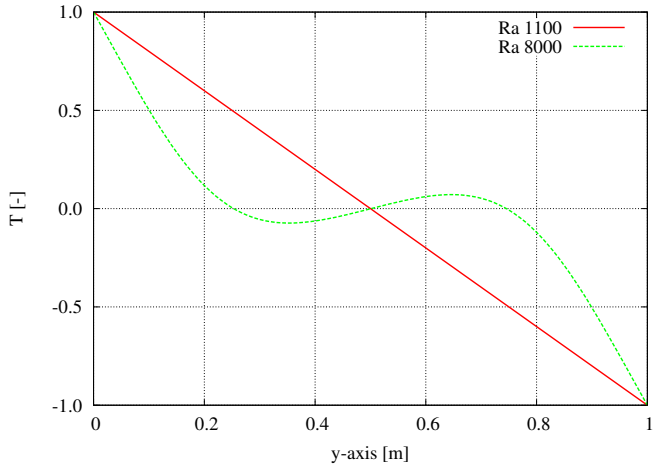


Figure 6.25.: Temperature plot through the domain at $x = 0.69$ (centre of the left-hand Bénard-cell in Figure 6.24(c)) for two different Rayleigh numbers.

motion and, thus, no velocity) and thermal transport is based purely on conduction. The isotherms have equidistant spacing and the temperature decreases linearly from T_{hot} to T_{cold}, which is in excellent agreement with the theory and can be seen in Figure 6.25.

For super-critical Rayleigh numbers, convection effects dominate and the fluid is in motion, forming two counter-rotating 2D rolls, called *Bénard-cells*. The temperature distribution is not linear anymore and can be seen in Figure 6.24(d). It should be noted here that in order to compare both results, the same colour scale was chosen for the velocity profile for $Ra = 1{,}100$ and $Ra = 8{,}000$, ranging from [0,0.5]. Hence, as in the sub-critical case, there is no motion, the complete domain is displayed in a blue colour, coding the velocity magnitude as zero.

In order to determine the critical Rayleigh number using the simulation results, the kinetic energy in the system is evaluated for several simulations using different Rayleigh numbers. In physics, the kinetic energy is computed by $E_{kin} = 1/2 \cdot m \cdot v^2$, where m represents the mass and v the velocity of the system. By expressing the mass m by the density ρ and the volume V, the kinetic energy can be rewritten as $E_{kin} = 1/2 \cdot \rho \cdot V \cdot v^2$. By evaluating the kinetic energy of every cell and summing up all the contributions, the total kinetic energy E_{kin} of the system at the time step t can be computed by

$$E_{kin}(t) = \frac{1}{2} \cdot \sum_i \sum_j \sum_k \left[\rho(i,j,k,t) \cdot V(i,j,k,t) \cdot \left(u_x^2(i,j,k,t) + u_y^2(i,j,k,t) + u_z^2(i,j,k,t) \right) \right] . \tag{6.16}$$

In order to achieve fast convergence towards the quasi-stationary solutions, all simulations used as their starting point the quasi-stationary solution of a simulation performed for $Ra = 8{,}000$, which is definitely higher than the critical Rayleigh number Ra_{crit}.

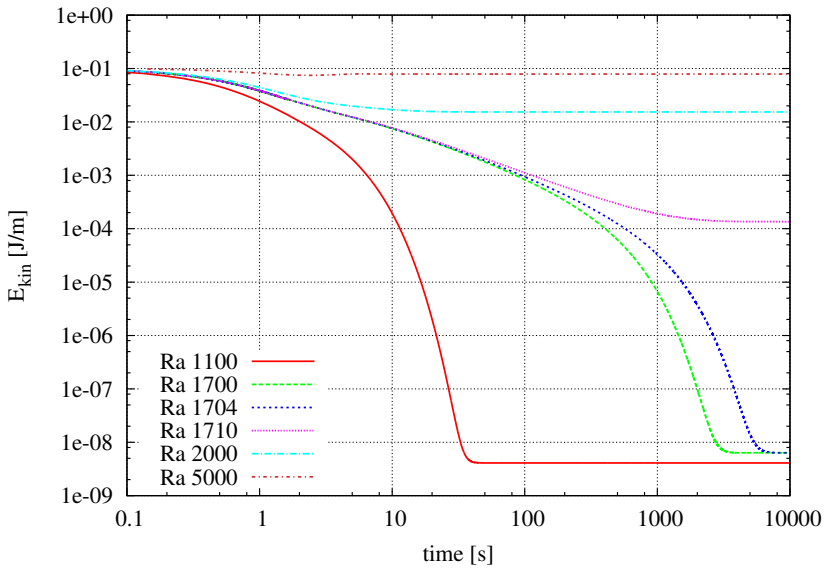
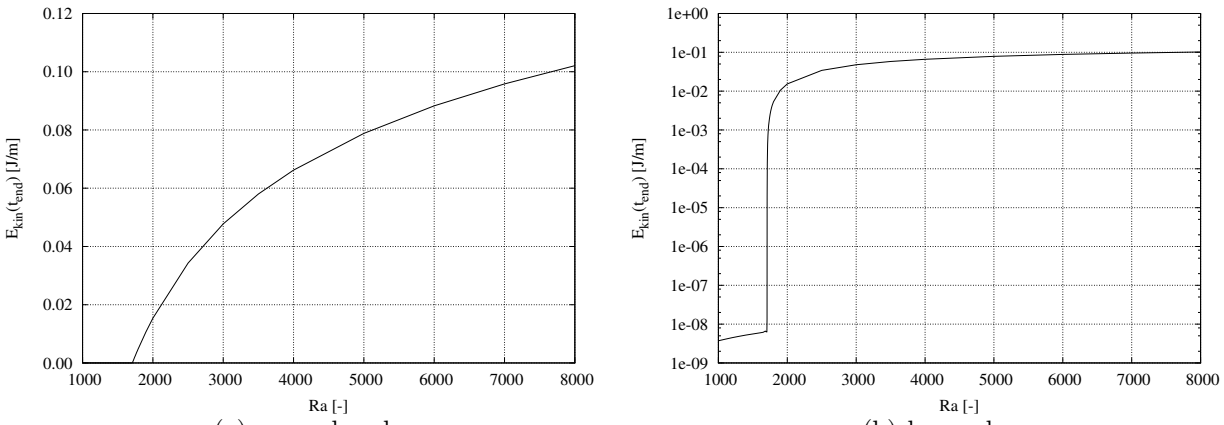


Figure 6.26.: Time evolution of the total kinetic energy in the system for different Rayleigh numbers.

Figure 6.26 shows exemplary for a few selected Rayleigh numbers the evolution of the total kinetic energy in $[\mathrm{J/m}]^2$ in the system during a simulation run when starting with the end state of $Ra = 8{,}000$. When $Ra \ll Ra_{crit}$ the system converges fast to a low level of kinetic energy, and for $Ra \gg Ra_{crit}$ to a higher final level of kinetic energy. For $Ra \approx Ra_{crit}$, i. e. the Rayleigh number is close to the critical value, the system needs much more time to converge to a certain level of kinetic energy.



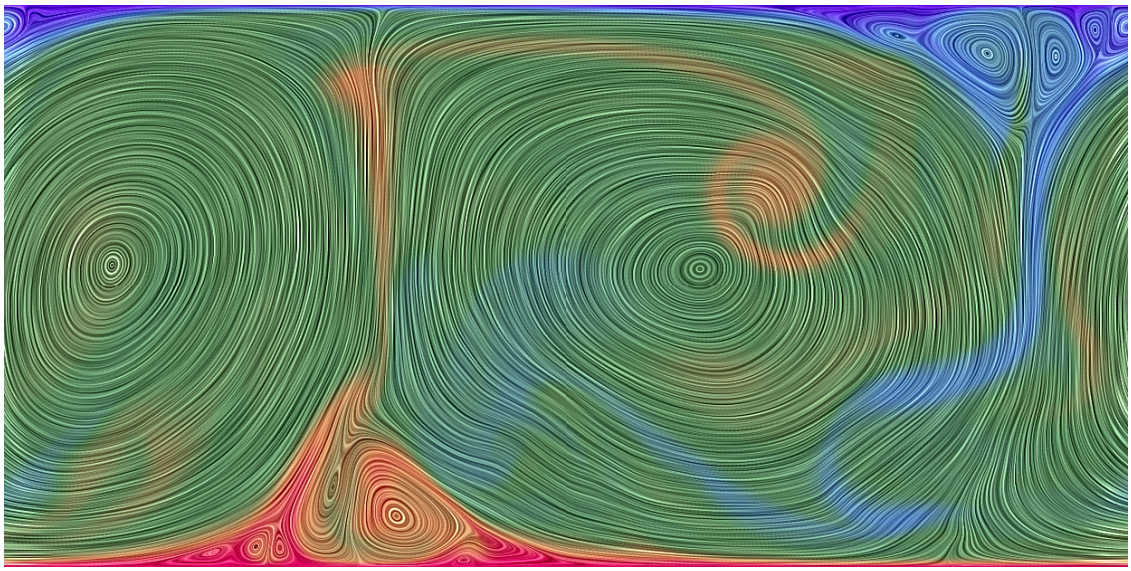
(a) normal scale (b) log-scale

Figure 6.27.: Total kinetic energy in the system at a quasi-stationary state at t_{end} for different Rayleigh numbers.

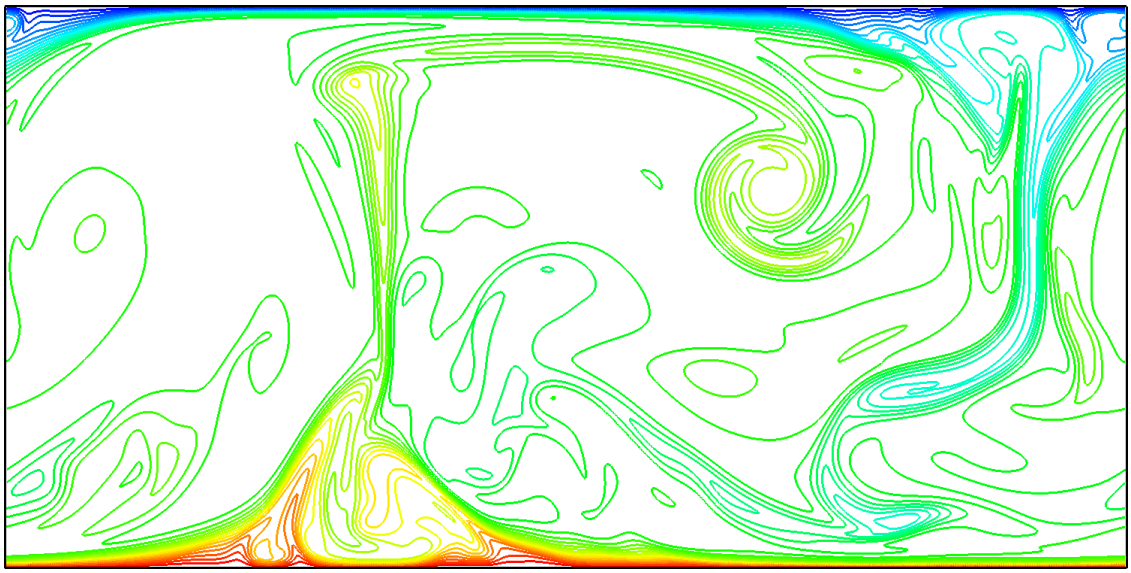
Figure 6.27 shows the evolution of the total kinetic energy in the system for different Rayleigh numbers derived at the end ($t > 5{,}000$ s) of Figure 6.26, plotted over the Rayleigh number

[2] as this is a 2D computation, the kinetic energy is in Joules per metres depth

itself. While Figure 6.27(a) shows the increase of the kinetic energy with increasing Rayleigh number, the sudden increase in the kinetic energy between $Ra = 1{,}704$ and $Ra = 1{,}710$ can be seen exceptionally well in the logarithmic plot in Figure 6.27(b). By interpolation, the critical Rayleigh number would be at approximately 1,707, which fits extremely well to the theoretical prediction of [PS40].



(a) velocity profile



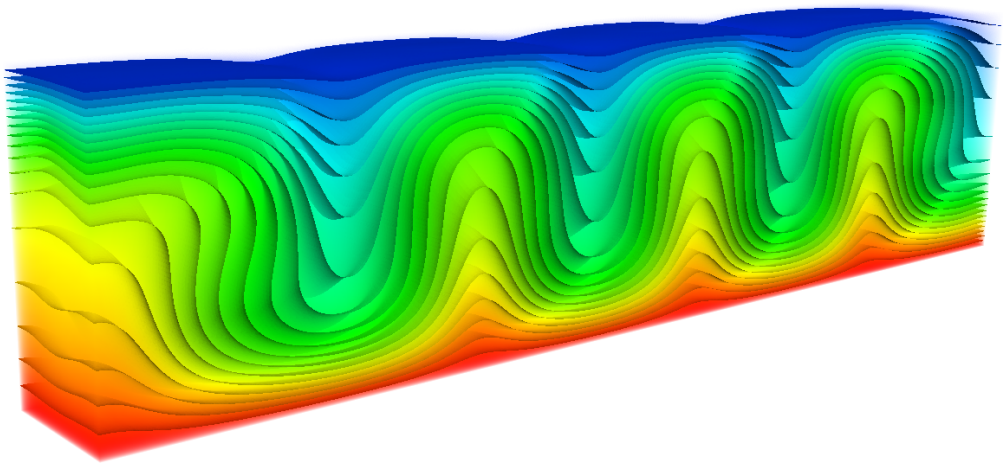
(b) isotherms

Figure 6.28.: Velocity and isothermal profiles for a 2:1 cavity with heated floor and cooled ceiling (Rayleigh-Bénard convection setup) at $Pr = 0.71$ for $Ra = 10^8$ computed on a $1{,}024 \times 512$ domain resolution.

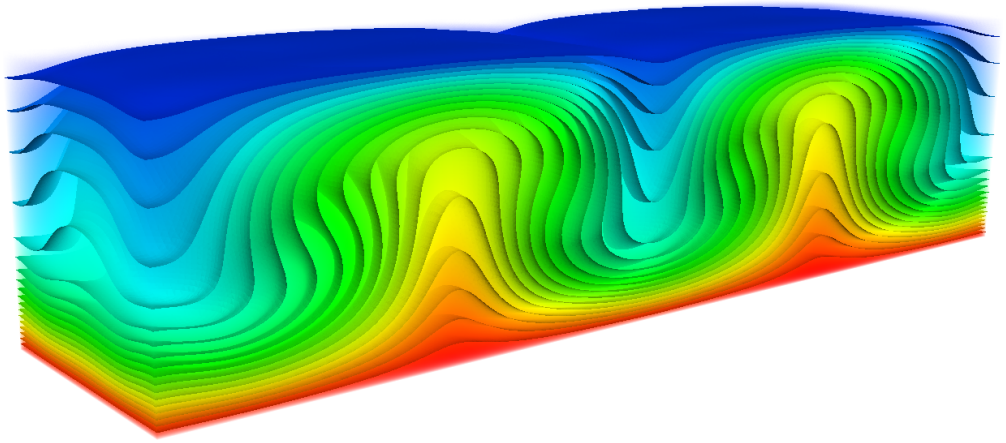
Figure 6.28 displays the same cavity using a Rayleigh number of 10^8, i. e. with fluid properties closer to real air. According to previous measurements, the absolute values will be off by

some percent, but the same fluid flow behaviour can be observed, which was reported in [vT04].

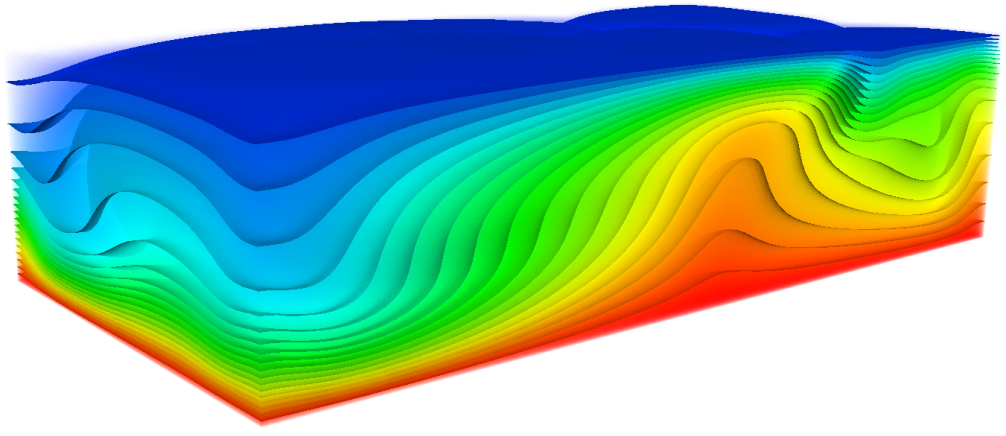
Figure 6.29 shows a 3D setup of three different domain configurations for the ratio 4:0.5:1 (Figure 6.29(a)), 4:1:1 (Figure 6.29(b)), and 4:2:1 (Figure 6.29(c)). The bottom wall was heated with $T = +1$ and the top wall was cooled with $T = -1$. All other walls were set as adiabatic, meaning perfectly insulated and, thus, not an infinite domain during this setup. All other conditions and settings were similar to the 2D scenario and $Ra = 30{,}000$ was chosen. The domain was computed uniformly up to depth 3 with different refinement strategies, depending on the ratio of the containment. The 3D effects can be clearly seen in the number of Bénard-cells which evolve in the domain. While 4:0.5:1 is still similar to the 2D case, the 4:2:1 scenario has a completely different flow behaviour with respect to the number of convection cells.



(a) Domain ratio 4:0.5:1



(b) Domain ratio 4:1:1



(c) Domain ratio 4:2:1

Figure 6.29.: 3D computation of the Rayleigh-Bénard convection for different domain ratios for $Ra = 30{,}000$ and adiabatic boundary conditions for the side walls. The temperature contours are colour coded from cold (blue) to hot (red).

7. Examples and Applications

In order to highlight the numerous advantages the previously presented code can offer apart from excellent scalability on large machines, two examples from engineering practice are presented in the following section, which offer higher complexity in terms of geometry as well as boundary conditions. For the sake of generating accurate initial conditions, a brief excursion into current engineering practice as well as the basics of thermal comfort assessment are presented first.

7.1. Current Engineering Practice in the Field of Building Performance Simulation

As already mentioned in Chapter 1, even nowadays the most powerful supercomputers are not able to compute a fully refined, thermally coupled CFD simulation over a whole year for a standard room, not to mention a complete building, using an adequate accuracy in a reasonable amount of time. Thus, different engineering models have to be used in order to evaluate the energy requirements in buildings or the temperature distribution in rooms over a time period of many years.

This is done in practice on multiple scales ranging from steady-state heat balances on a coarse level, dynamic single- or multi-zone models, to detailed CFD models, into which the current code can be classified. A very detailed description as well as a historical overview is given in [vT10] and a very short wrap up is presented in the following paragraphs.

For the steady-state heat balances, the energy consumption of buildings is evaluated on a monthly basis and energy demands for heating and cooling are calculated approximately in order to compensate for heat gains or losses due to solar irradiation or air infiltration, for example. The heat storage capacity is not represented accurately, but can be estimated by using lump-sum factors given in standards, for example. This whole procedure could be done by hand, but usually a spreadsheet is used. The methods and/or computer programs applied can already provide a good estimate about the computational effort this method requires. Hence, it is a very coarse method in space as well as in time, but delivers fast estimates of the required energy demands for the heating and/or cooling of buildings in early design stages, when not a lot of information is available in any case.

In order to capture a more detailed picture, either the temporal or spatial dimension can be refined. For a whole year simulation of a room or a complete building, the temporal

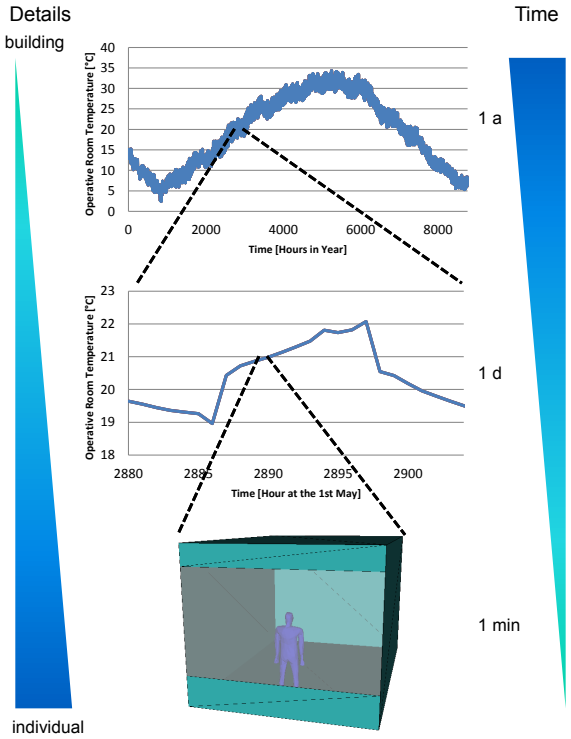


Figure 7.1.: Temporal and spatial discretisations for building performance simulation (cf. [vTFER09]).

dimension is refined in the range of seconds to hours and the resulting models are called *zonal models*. Figure 7.1 shows a multi-scale-based approach with different levels of detail. A complete building can be analysed over a whole year, single rooms are regarded in more details on a finer scale, and a single individual can be analysed in detail for a few seconds to minutes.

Thermal single- or multi-zone models are usually based on anisotropic finite volume models [Cla01]. They have still a coarse spatial refinement, but the building mass is modelled correctly with specific wall, floor, and ceiling construction parameters, for example. The thermodynamic conservation equations are solved in every time step while taking the influence of boundary conditions such as weather parameters, solar irradiation, internal heat sources, and specific usage patterns of the facility management into account. If multiple connected zones are present, the mass flow rates in between the zones have to be evaluated as well, using the Bernoulli[1] equations, for example. Theoretically, by reducing the zone sizes to the range of centimetres or less, an equation formulation equivalent to a CFD formulation results and the transition from zonal approach to CFD approach is blurred and depends on definitions. In the standard case, however, the zones have still a considerable size and can deliver results for whole year simulations in the order of one hour or less computation time depending on the degree of spatial and temporal refinement.

A lot of different computer programs are available nowadays to tackle a simulation according

[1]The Bernoulli equations are a simplification of the Euler equations (2.21) presented in Section 2.3.

to a zonal model setup. A good overview of existing software and their history can be found in [vT10]. For the purpose of this work, a self-written solution for a single-zone thermal computation is used, where the most important features are integrated, such as long- and short-wave irradiation, 'outside' climate conditions, or internal heating and cooling according to facility management strategies of occupancy. Validations according to [VDI6020] were performed for the code given and presented in [vTFER09]. Further, detailed descriptions can also be found in [vT10].

A more accurate picture can be obtained by refinement of the temporal and the spatial dimension to milliseconds and millimetres, respectively. Using a detailed CFD simulation, the velocity pattern can be evaluated for a very limited time range quite accurately, given that the adequate boundary conditions can be applied for the CFD approach.

From the previously applied method (either zonal or CFD approach), the temperature distribution is now known for the given room or building. A post-processing of these values can deliver a lot of additional information for occupants, such as a thermal comfort assessment, which is discussed in the next section.

7.2. Thermal Comfort Assessment

Nowadays, indoor climate assessment is of more and more importance in early building design phases as well as in the later operation of the aforementioned. It has a major impact on health, productivity, and the thermal comfort of the occupants (cf. [dDB02], [Sto91]).

Fanger [Fan70] was one of the first pioneers in the field of thermal comfort assessment. He analysed the heat balance of the human body and related it by statistical means to predicted mean votes (PMV) of people's satisfaction with the thermal environment. The PMV is expressed on a seven-point scale (also called ASHRAE thermal sensation scale) ranging from -3 (very cold) to +3 (very hot). The PMV values can be converted into a predicted percentage of dissatisfied (PPD), which should be kept as low as possible inside of buildings. Several international and national standards such as [ISO7730], [EN15251], or [ASH55] were built on top of this predicted mean vote and define so-called comfort zones for different classes of buildings. Furthermore, the models in the standards take clothing insulation or air humidity into account.

The global model of Fanger relates the net heat production to the sum of skin heat losses, respirative losses, and heat storage. It considers the body as a whole and delivers one overall value which can be compared to predefined limits from standards, for example. Another approach is to refine the equation into two thermal compartments, skin and core, resulting in a two-node model as proposed by Gagge [Gag73] or a multi-segment model as proposed by Stolwijk [Sto71].

Detailed human thermoregulation models such as proposed by Fiala (cf. [FLS99], [FLS01]) take reactions of the central nervous system into account which are triggered by multiple

signals from core and peripheral sensors. Significant indicators are the mean skin temperature $T_{sk,m}$, the mean skin temperature variation over time $\partial T_{sk,m}/\partial t$, and the hypothalamus core temperature T_{hy}. The central nervous system can actively react to the surrounding thermal conditions with active measures in order to compensate a heat surplus or deficit. Therefore, the skin blood flow can be changed by vasoconstriction and vasodilatation. Evaporative heat loss can be achieved by activating the sweating mechanism and shivering can produce heat by increasing the metabolism in the muscles. Detailed comparisons to other models and numerical results are presented in his work.

Fiala's thermoregulation model was implemented in the software suite THESEUS-FE from P+Z Engineering[2]. *THESEUS-FE is a professional software tool for steady-state and fully transient thermal management applications like complete cabin interior analysis including passenger comfort studies. It is designed for use in automotive, transportation and aerospace industries mainly but is applicable for other markets, too.*[3] Detailed information about the thermal manikin model and the numerical treatment using a finite element-based numerical discretisation can be found in [PWS07].

In [Fri08] and [FvT09] a virtual climate chamber concept is presented, which consists basically of a GUI connected via a back-end interface to the internal computation routines of THESEUS-FE. Figure 7.2 shows an exemplary use of the virtual climate chamber with synthetic boundary conditions and the corresponding reactions of the human manikin over time. The ambient temperature T_{amb} as well as the clothing insulation was changed at specific points in time (see diagrams *a*) and *b*)). The resulting mean skin ($T_{sk,m}$) and mean surface temperatures ($T_{sf,m}$) are plotted in part *a*). Part *c*) shows the global predicted mean vote according to Fanger and the dynamic thermal sensation (DTS) according to Fiala on a seven-point ASHRAE scale. The DTS can give detailed insights into the dynamic reactions of the human system, although it is still evaluated on a global scale. A more detailed insight is given by local thermal sensation votes (LTSV) described in detail in [vTFP+09]. The LTSV values are plotted in part *d*) for three specific parts of the body. These show experimental correlations determined by the Fraunhofer Institute for Building Physics in Holzkirchen, Germany, for specific body parts. Part *e*) shows the LTSV values for the whole body with colour coding (deep blue for -3 and dark red for +3) for different simulation times. Using this detailed method, predictions can be made for different parts of the body.

Furthermore, [vTFP+09] describes coupling strategies of the previously mentioned interface to zonal models, for example. These presented concepts will now be extended by the CFD program from this work, and an overall interaction of all the components is shown exemplarily by simulating a test room containing a human manikin.

[2] `http://www.puz.de`
[3] Source: `http://www.theseus-fe.com`

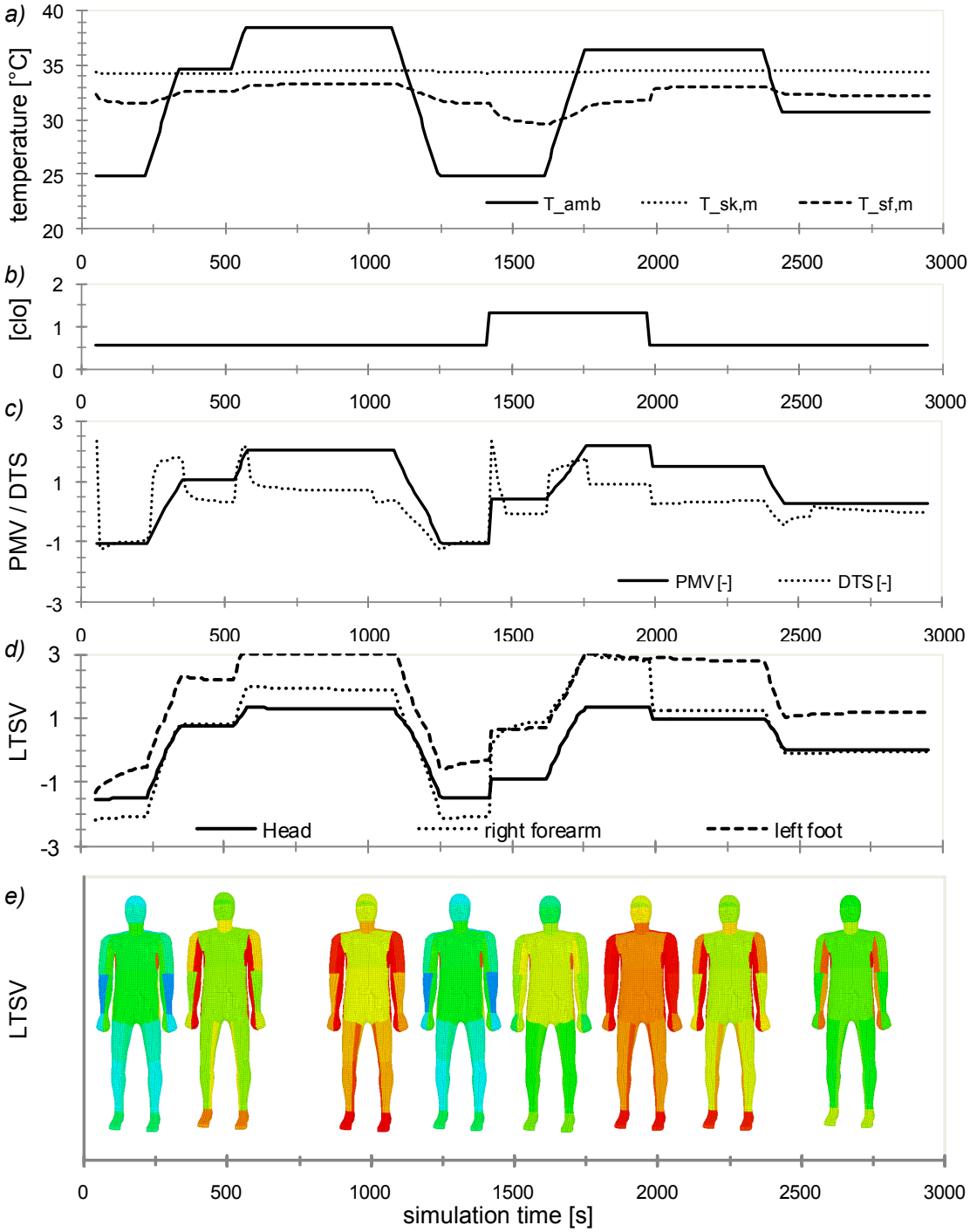


Figure 7.2.: Exemplary result of the virtual climate chamber application showing boundary conditions for T_{amb} as well as clothing insulation in *clo*. Results for the skin and surface temperature are given as well as the predicted mean vote (PMV) and the dynamic thermal sensation index (DTS) according to Fiala. Furthermore, the local thermal sensation value (LTSV) introduced by the Fraunhofer Institute for Building Physics (IBP) is indicated as well. (cf. [FvT09])

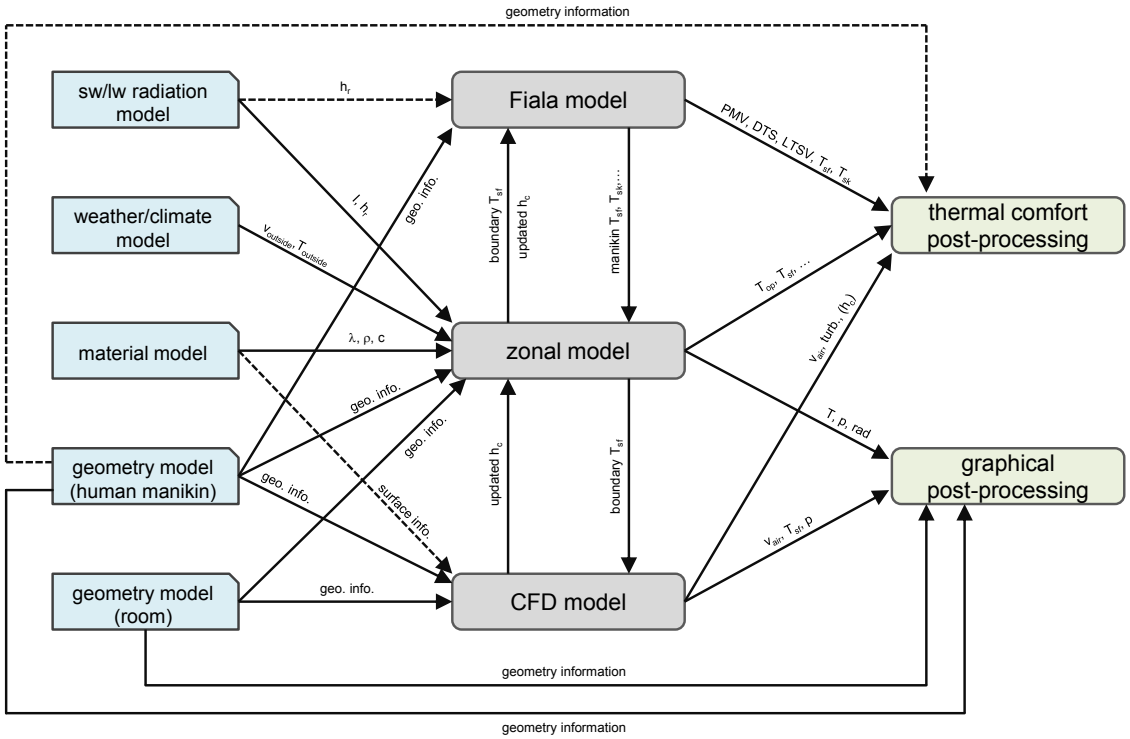


Figure 7.3.: Coupling concept used for simulation of human manikin in a test room.

7.3. Test Example – Human in Climate Chamber

The first complex example which is tackled here consists of a human manikin in the middle of a test room with a geometric setup corresponding to the room from [VDI6020], which is 5 m in length, 3.5 m in width, and 3 m in height. The front surface (at $x = 0$) has a window spanning the complete width of the room with a surface of 7 m^2 and is exposed to outside weather conditions specified in the test reference year (TRY) data for region 5 (Franconia and northern Baden-Württemberg, station number 10 655). All other walls are connected adiabatically at the outer wall construction hull, whose material composition is specified in detail in Appendix A of [VDI6020]. The corresponding room setup can be seen in the lower part of Figure 7.1. The numerical manikin represents a standard male test subject with a height of 1.70 m.

A coupled multi-scale and multi-physics simulation is performed. The temporal multi-scale simulation couples a zonal computation over a whole year to a very detailed CFD simulation running for up to one minute, delivering a velocity profile and convective heat transfer coefficients back to the zonal simulation. Furthermore, a multi-physics aspect is present, as Fiala's human thermoregulation model is coupled to the zonal simulation and the CFD simulation at the same time. This setup can be seen in Figure 7.3, which depicts all the relations and exchanged quantities in detail.

A geometry model, containing the surrounding room and the human manikin, is set up together with a material model for the room, a weather and climate model (basically a TRY

database), and a radiation model for long- and shortwave irradiation computation. All these components feed specific information into either the zonal model, the CFD model, or the thermoregulation model. These three main components are coupled and forward updated information back and forth during the simulation runs. Depending on the physical background, the exchange does not need to happen at every time step, as the thermoregulation model reacts much more slowly to environmental changes than the CFD velocity profile in the room, for example.

The zonal simulation is the central component and simulates a time span of a whole year. The thermoregulation model is tightly coupled to the zonal model, evaluating the responses of the human manikin with respect to surface temperatures and thermal comfort assessment over the complete year. At a predefined time step of 6,036 year-hours (8th September at noon), a CFD 'snapshot' is computed, using the boundary conditions from the zonal model as well as the surface temperatures of every sector of the Fiala model as the initial conditions.

Table 7.1.: Fluid flow properties used for the CFD simulation of the test room for natural convection scenarios.

Ra	=	10^7	[-]
Pr	=	0.71	[-]
ν	=	$4.6433 \cdot 10^{-4}$	[m^2/s]
α	=	$6.5398 \cdot 10^{-4}$	[m^2/s]
k	=	0.7927786	[W/(m·K)]
β	=	$3.4 \cdot 10^{-3}$	[1/K]
ρ_∞	=	1.205	[kg/m^3]
c_p	=	1,006	[J/(kg·K)]
T_∞	=	294.66	[K]
$T_{hot} - T_{cold}$	=	18.54	[K]
$\vec{g}$	=	$(0, 0, -9.806)^T$	[m/s^2]
L_{char}	=	1.7	[m]

Table 7.1 shows the fluid properties applied for the given room setup. Basically, a computation using a Rayleigh number of 10^7 is performed, thus simulating not real air, but an arbitrary liquid slightly more viscous than air[4]. Nevertheless, the values are close enough to obtain quantitative results. These values were chosen here as, in the verification examples, a similar setup proved to deliver reliable results. The numerical discretisation on the CFD side uses a bisection in every direction up to depth 4 and a block size of (12,16,12), leading to a domain having 4,681 l-grids and a total of 10.8 million cells.

The wall temperatures are fixed to surface values delivered from the zonal model at the interface of wall to air, and range from 17.10 °C to 18.44 °C depending on the wall selected. No-slip boundary conditions with fixed Dirichlet boundary conditions for the temperatures are applied.

[4]According to [Bej13] Appendix D, air at 20°C has $\nu = 0.150 \cdot 10^{-4}$ m^2/s.

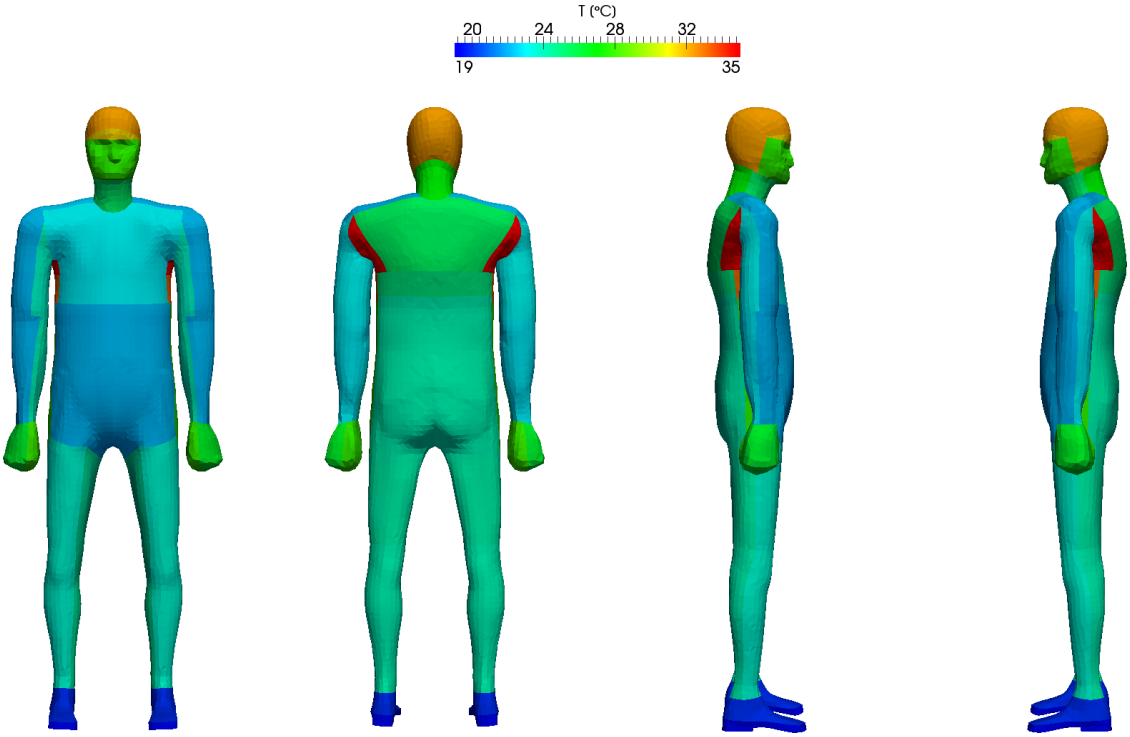


Figure 7.4.: Temperature distribution on the surface of the numerical manikin wearing winter clothing with an insulation factor of 1.33 clo at simulation year hour 6,036.

The surface temperatures for the human manikin are fixed in every sector of the model by applying results from the thermoregulation model. Figure 7.4 shows the irregular initial temperature distribution on the surface of the manikin which is 'insulated' with winter clothing. The head has an elevated surface temperature, as no insulation is present and a lot of metabolic heat is produced at this specific place according to physiological measurements represented in Fiala's modelling.

Figure 7.5 shows the numerically determined values for a cutline starting from the head of the manikin (at $z = 0$) and going to the ceiling of the test room (at $z = 1.3$ m over the head of the manikin). Furthermore, it shows an interpolated polynomial function of fourth order valid in the region $z \in [0, 0.15]$ m, which corresponds to the area of interest for the convective heat exchange with the manikin's head. The interpolated function can be determined as

$$T(x,y,z)|_{x=2.5,y=1.75} = -52{,}852\ z^4 + 13{,}760\ z^3 - 921.31\ z^2 - 39.498\ z + 32.683\ [^\circ\mathrm{C}] \quad , \quad (7.1)$$

and has a correlation coefficient $R^2 = 0.9952$ in its region of definition. The figure also shows that, outside the predefined region, the interpolation is invalid.

The derivative of the interpolated function $T(x, y, z)$ can be determined analytically and evaluated at the position $z = 0$ in order to compute the convective heat transfer coefficient h_c according to [Bej13] as

$$h_c = \frac{-k \left.\frac{\partial T(x,y,z)}{\partial z}\right|_{z=0}}{T_0 - T_\infty} \quad , \quad (7.2)$$

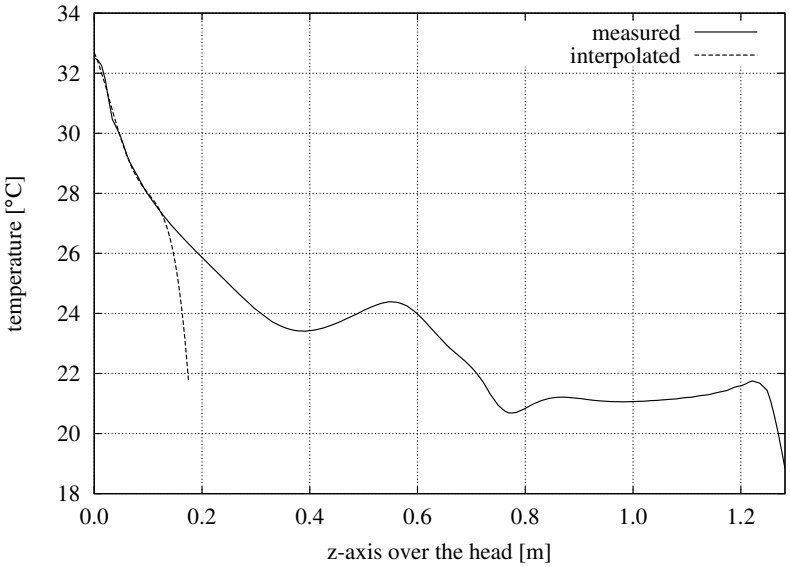


Figure 7.5.: Temperature distribution in °C in the z-direction starting at the top of the head of the manikin until the ceiling is reached. The interpolated polynomial function of fourth order is valid in the range $[0, 0.15]$ m.

where k is the heat conduction coefficient in [W/(m·K)], T_0 the temperature of the surface in [K] (in this case the temperature of the manikin's head), and T_∞ the temperature of the fluid at rest in [K]. Thus, for this specific problem setup and simulation results, the heat transfer coefficient for the manikin's head can be computed[5] as

$$h_{c,sim} = \frac{-0.7927786 \cdot (-39.498)}{32.683 - 21.66} = 2.841 \ \frac{\mathrm{W}}{\mathrm{m}^2 \cdot \mathrm{K}} \quad . \tag{7.3}$$

Experimental investigations made by [KTI+08] using a regression analysis for a standing person in a room yielded a convective heat transfer coefficient of

$$h_{c,exp} = 1.007 \cdot (T_0 - T_\infty)^{0.408} \quad , \tag{7.4}$$

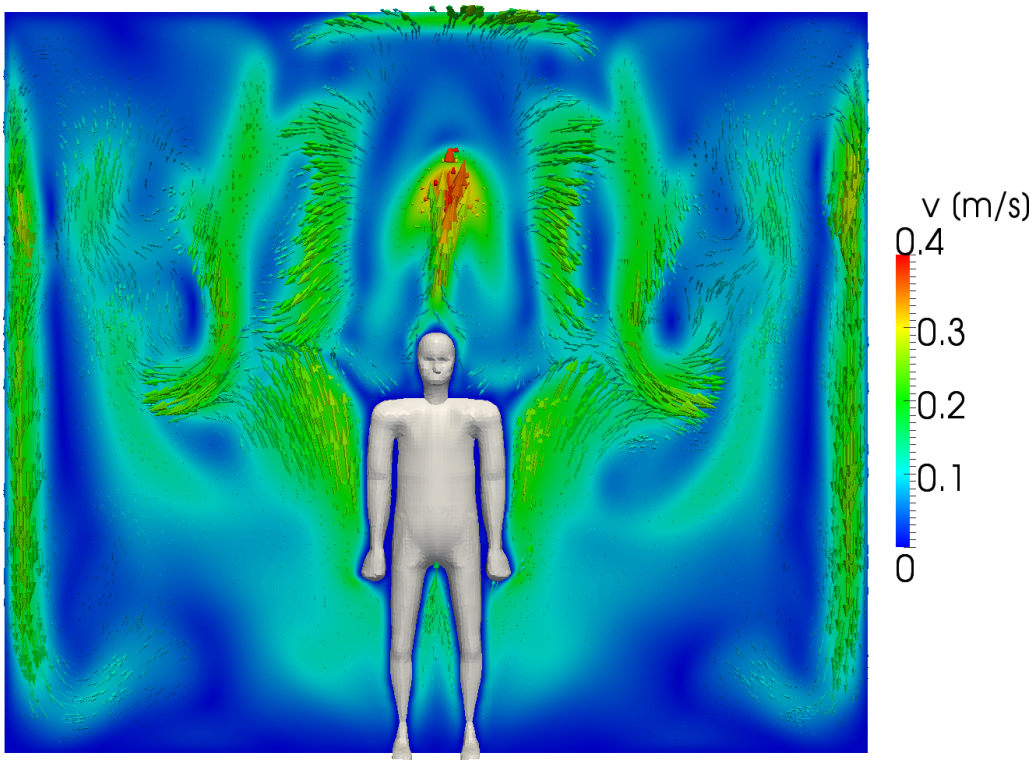
which results in an experimentally determined convective heat transfer coefficient of

$$h_{c,exp} = 1.007 \cdot (32.683 - 21.66)^{0.408} = 2.681 \ \frac{\mathrm{W}}{\mathrm{m}^2 \cdot \mathrm{K}} \quad , \tag{7.5}$$

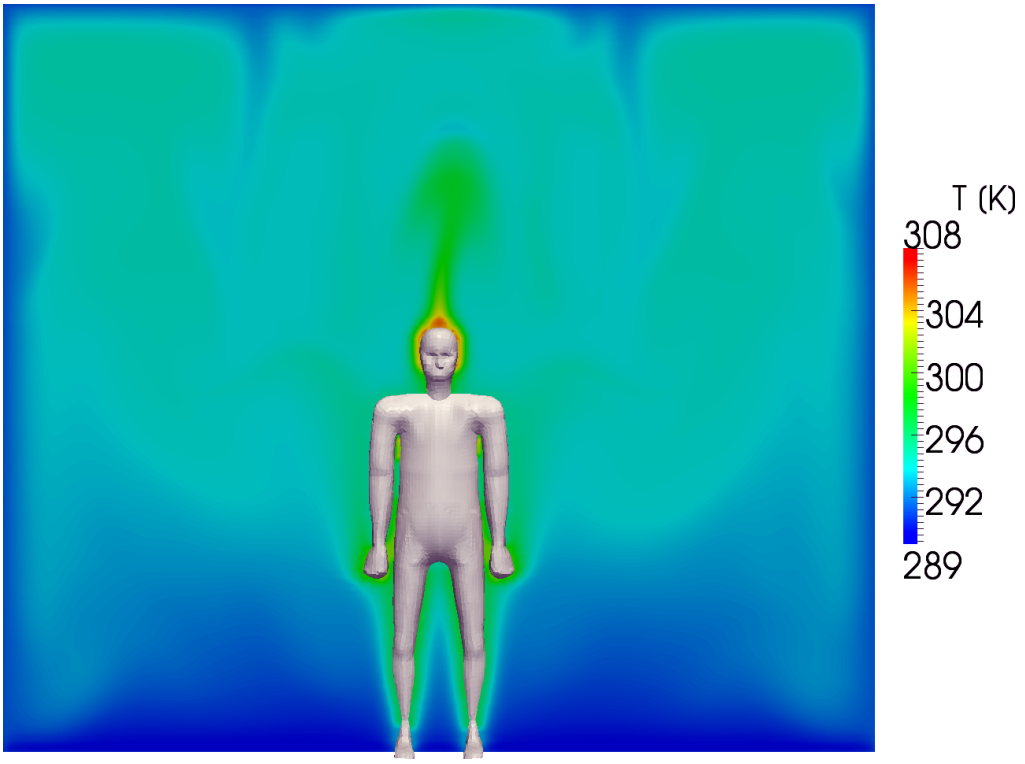
and corresponds very well to the results obtained, keeping in mind that the simulation was performed for $\nu = 4.6433 \cdot 10^{-4}$ [m^2/s] ($Ra = 10^7$) instead of real air, which would have $\nu = 0.150 \cdot 10^{-4}$ [m^2/s]. Furthermore, no turbulence model is applied in the simulation at the moment. Thus, the $h_{c,sim}$ value has to be seen as an integrated value over several time steps, which would effectively eliminate the turbulent fluctuations (see also Section 2.6).

Figure 7.6 shows a velocity and a temperature profile on a cut-plane at $x = 2.5$ m in the

[5] As only temperature differences occur here, [°C] is equivalent to [K]. All simulations were performed in [K] and the values were converted to [°C] for an easier interpretation.

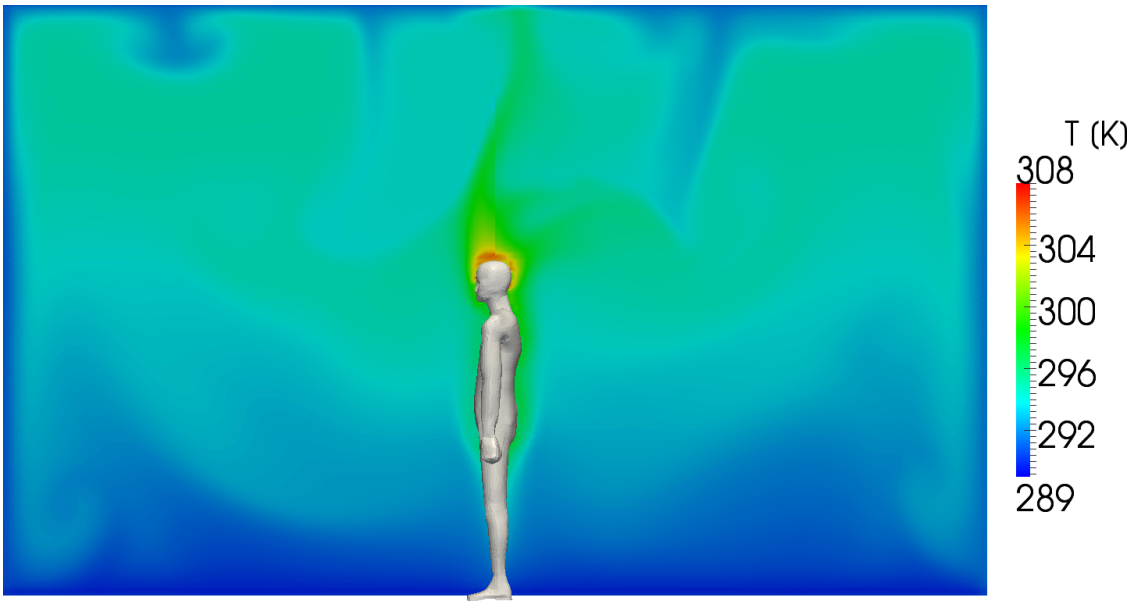


(a) velocity magnitude in cut-plane at $x = 2.5$ m

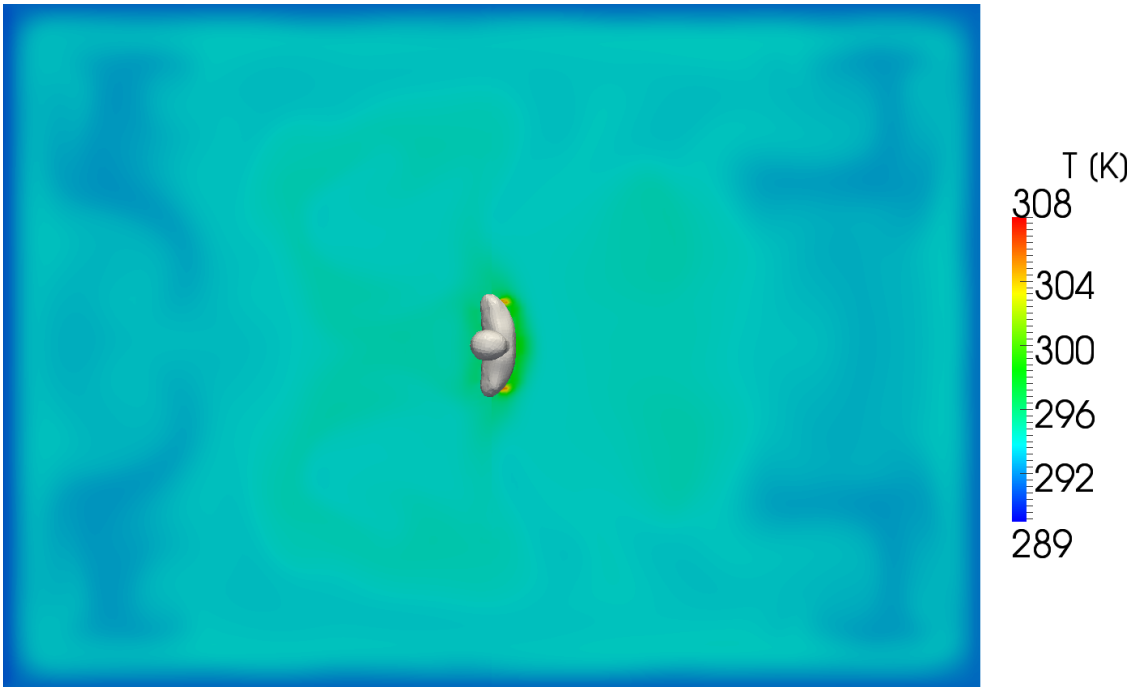


(b) temperature in cut-plane at $x = 2.5$ m

Figure 7.6.: Velocity and temperature profiles through a cut-plane at $x = 2.5$ m for the test room computed using $Ra = 10^7$.



(a) temperature in cut-plane at $y = 1.75$ m



(b) temperature in cut-plane at $z = 1.5$ m

Figure 7.7.: Temperature profiles through a given cut plane for the test room computed using $Ra = 10^7$.

middle of the room. The velocity profile is irregular due to the (irregular) manikin standing in the middle of room. Nevertheless, the typical characteristics of a hot plume rising over the head of the manikin can be observed.

Similar measurements were performed by van Treeck et al. for a sitting manikin in a climate chamber using experimental particle image velocimetry (PIV) measurements at the Fraunhofer Institute for Building Physics IBP in Holzkirchen. Details about the setup and the results can be found in [vTM13]. The velocity and temperature profiles are shown in 2D cuts, as the PIV measurements can unfortunately capture only 2D slices because the method applied involves photographing tracer particles in a fluid illuminated in a 2D laser plane.

Figure 7.7 shows the temperature distribution on a cut-plane at $y = 1.75$ m (middle of the room) and $z = 1.5$ m height over the floor (approximately a bit lower than shoulder height of the manikin). The 3D effects of the fluid motion can be observed very well here. Furthermore, effects such as falling plumes of colder air can be seen in Figures 7.6(b) and 7.7(a), which correlate to the 2D computations of the Rayleigh-Bénard instability shown in Section 6.5.

Using these results, further thermal post-processing can be performed in order to determine the PMV or the percentage of dissatisfied people due to local draughts or thermal imbalances, for example.

7.4. Test Example – Operating Theatre

As a second quite complex example, an operating theatre already dealt with in the dissertation of Wenisch [Wen08] and Pfaffinger [Pfa12] is analysed. Until now, both dissertations computed the room using a Lattice-Boltzmann approach on a uniform l-grid in an isothermal setting only. The underlying geometry is based on one of the special operating theatres located at the university hospital 'Klinikum rechts der Isar' in Munich with the geometric dimensions of $6.3 \times 6.25 \times 3.5$ m and using special ventilation systems. Air streams in over one complete side wall of the room and exits through an open door on the opposite side. The geometry setup is depicted in Figure 7.8 and contains 84,072 triangular surfaces for describing the interior obstacles in the room.

The fluid parameters applied are listed in Table 7.2. Although the viscosity is relatively high and will not represent 'real' air very accurately, it was applied in the following settings for testing the complex setup using a thermally coupled CFD computation in a mixed convection scenario, where the influence of the surgical lights on the fluid flow itself can be analysed.

As the two ceiling-mounted surgical lights get quite hot, they are assigned a high temperature T_{lamps}. All other inanimate objects are set to a cold temperature $T_{obstacles}$ and the manikins are assigned the temperature $T_{manikins}$. The inflow and outflow temperature boundary condition are set to adiabatic. All other wall temperatures are chosen as Dirichlet conditions

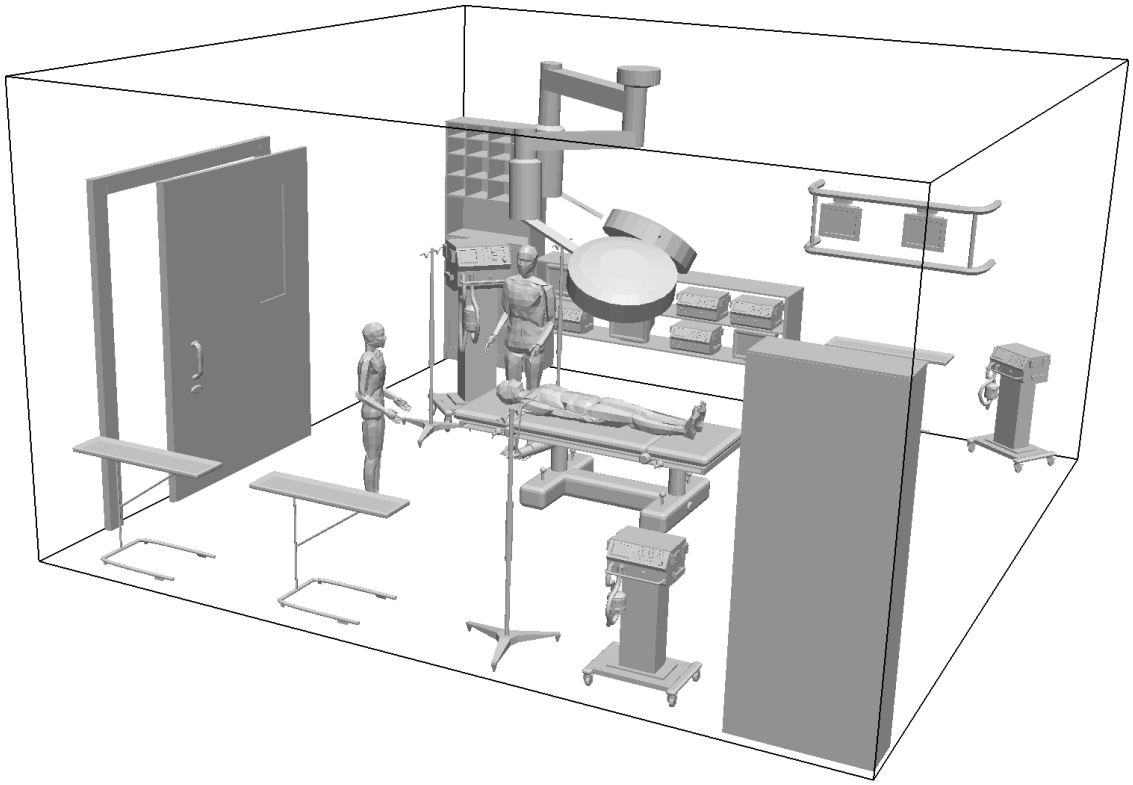


Figure 7.8.: Geometry setup of the operating theatre including instruments, patient, and surgeons.

Table 7.2.: Fluid flow properties used for the CFD simulation of the operating theatre for mixed convection scenarios.

Ra	=	10^6	[-]
Pr	=	0.71	[-]
ν	=	$5.94276 \cdot 10^{-3}$	[m^2/s]
α	=	$8.3701 \cdot 10^{-3}$	[m^2/s]
k	=	10.1465	[W/(m·K)]
β	=	$3.4 \cdot 10^{-3}$	[1/K]
ρ_∞	=	1.205	[kg/m^3]
c_p	=	1,006	[J/(kg·K)]
T_∞	=	294.66	[K]
$T_{hot} - T_{cold}$	=	34.5	[K]
T_{lamps}	=	324.66	[K]
$T_{obstacles}$	=	290.16	[K]
$T_{manikins}$	=	299.50	[K]
$\vec{g}$	=	$(0, 0, -9.806)^T$	[m/s^2]
L_{char}	=	3.51	[m]
u_{inflow}	=	0.05	[m/s]

in the range of 17.10 °C to 18.44 °C depending on the wall selected, exactly as in the room described in Section 7.3.

Furthermore, different computational resolutions were used in order to assess whether the discretisation can be coarsened without losing too much accuracy. A recursive bisection was chosen and all block sizes were set to 16×16×8. Together with the room dimensions, this will produce nearly quadratic cells which are preferable. Settings *adapt*3 and *adapt*4 represent an adaptive setup up to depth 3 or 4, respectively, and are depicted in Figure 7.9. Settings $unif3$ and $unif4$ represent a uniform setup up to the given depth. Table 7.3 contains the number of l-grids and cells present in the simulation for the different scenarios as well as the geometric dimensions of the smallest cells present in the computational domain.

Table 7.3.: Domain sizes for measurements using a recursive bisection and a block size of 16×16×8.

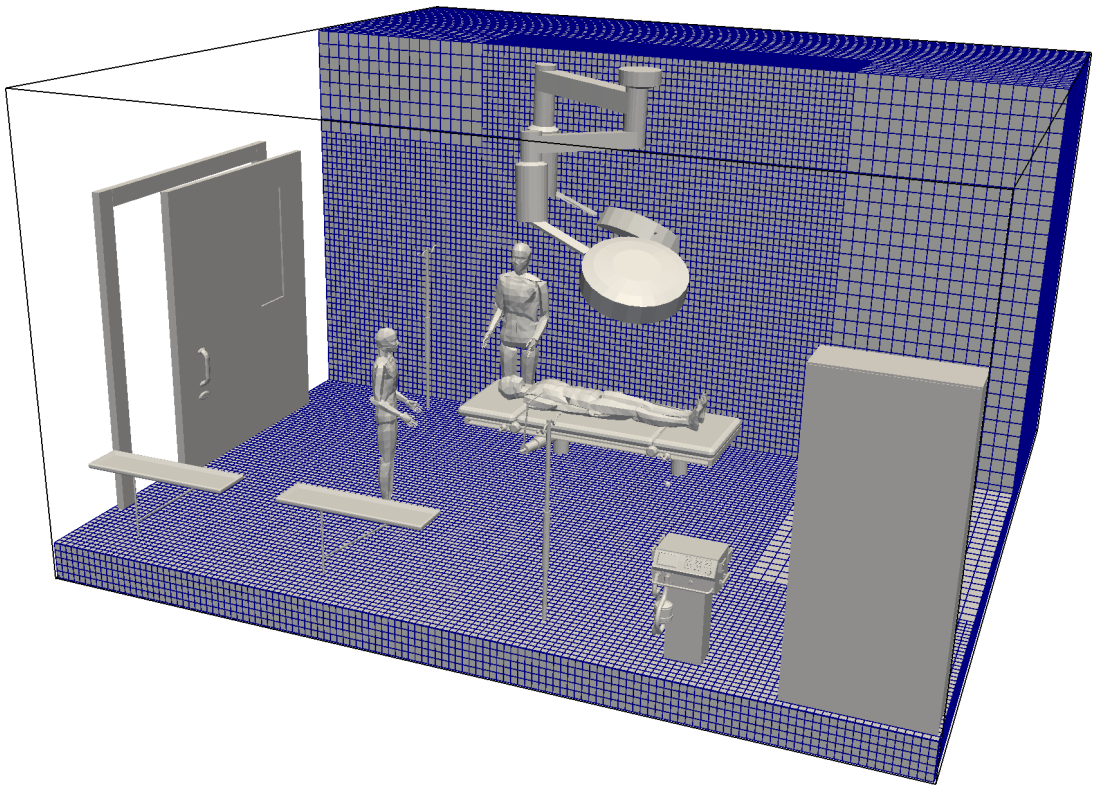
setting	number of l-grids	number of cells	smallest cell resol. [cm]
*adapt*3	441	$0.90{\cdot}10^6$	4.91×4.85×5.48
$unif3$	585	$1.20{\cdot}10^6$	4.91×4.85×5.48
*adapt*4	1,881	$3.85{\cdot}10^6$	2.46×2.43×2.74
$unif4$	4,681	$9.59{\cdot}10^6$	2.46×2.43×2.74

Figure 7.10 shows the temperature distribution in the operating theatre for two different adaptive setups. Parts of the domain containing simulation results have been set to transparent in order to visualise the adaptive geometry setup.

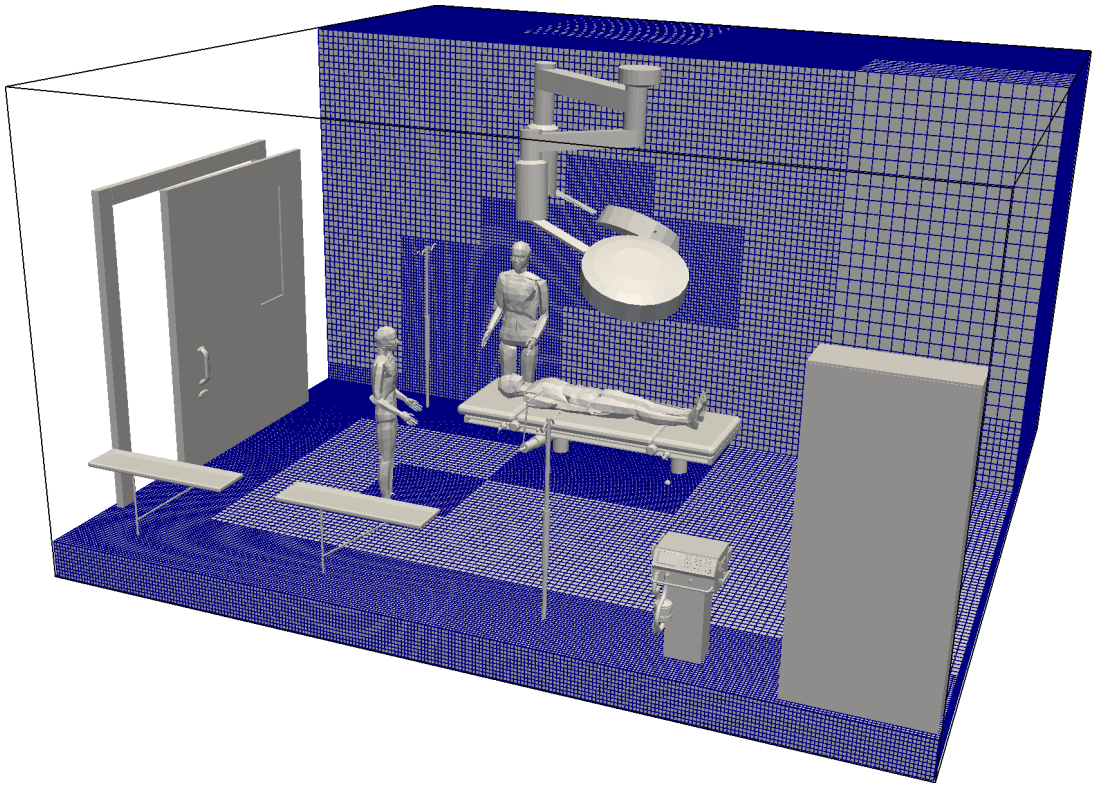
Visually, it can be clearly seen that the characteristics are very similar for the two different simulations. In order to quantify the differences, the total kinetic energy in the complete system was compared using Equation (6.16) and yielded a difference of 5.08% from setup *adapt*3 to *adapt*4. For the 'productive' run, the two setups were computed on the MAC cluster using different numbers of processes. In order to compare the actual run times, however, a small simulation using 16 processes was performed until 0.01 seconds simulation time. *adapt*3 finished in 17.12 seconds, whereas *adapt*4 needed 300.27 seconds, which is around 17.5 times slower than *adapt*3. This difference does not only result from the fact that *adapt*3 has approx. four times fewer cells than *adapt*4, but is also due to a reduced time step which is needed in order to guarantee a stable computation according to the CFL conditions stated in Section 3.7. The maximal time step was chosen in this case according to Equation (3.66) with $\tau = 0.1$.

Comparing the total kinetic energy for *adapt*4 and $unif4$, a difference of around 1.95% is measured. Keeping in mind that $unif4$ has 2.5 times more cells than *adapt*4, this difference is very acceptable. Furthermore, in order to compute 0.01 seconds using 16 processes on the MAC cluster, 529.56 seconds are necessary for $unif4$, which is 1.7 times slower than *adapt*4. Hence, an adaptive setup has huge advantages for such a complex scenario compared to a simple uniform setup.

Figures 7.11 and 7.12 show streamlines for the uniform setup on depth 3 after 100 seconds simulation time. In Figures 7.11(a), 7.11(b), and 7.12(a) 100 seed points were used, dis-

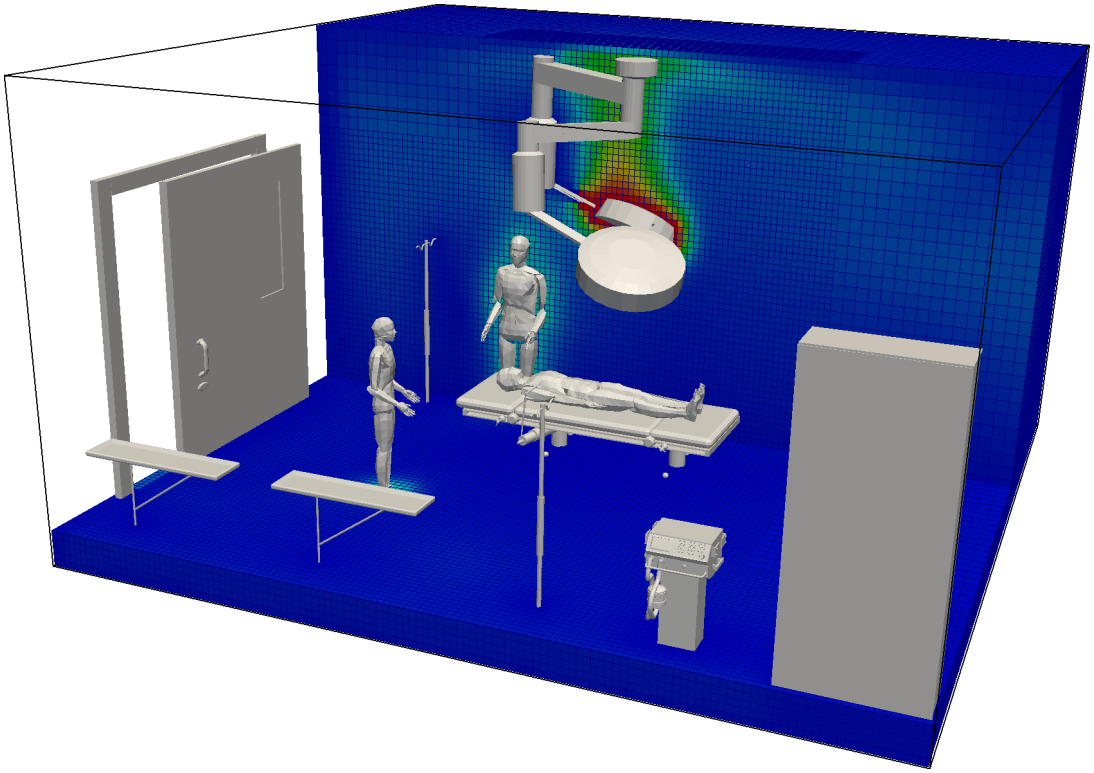


(a) *adapt*3 setup

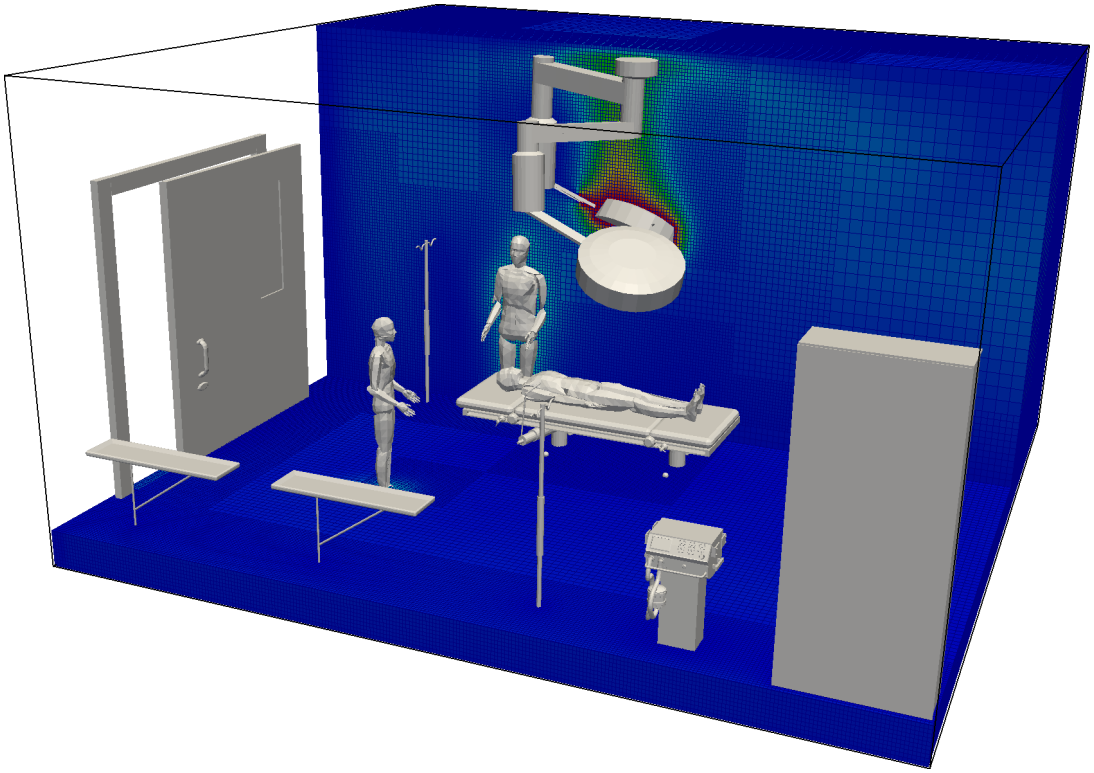


(b) *adapt*4 setup

Figure 7.9.: Adaptive l-grid refinement for the *adapt*3 and *adapt*4 setup.

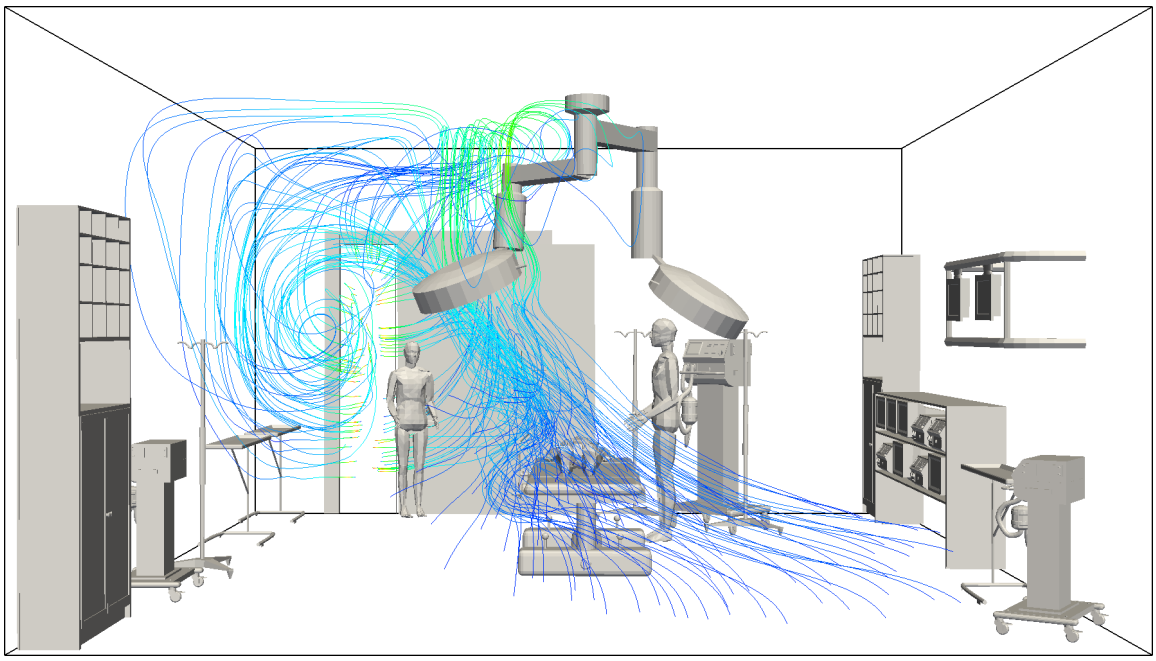


(a) *adapt*3 setup

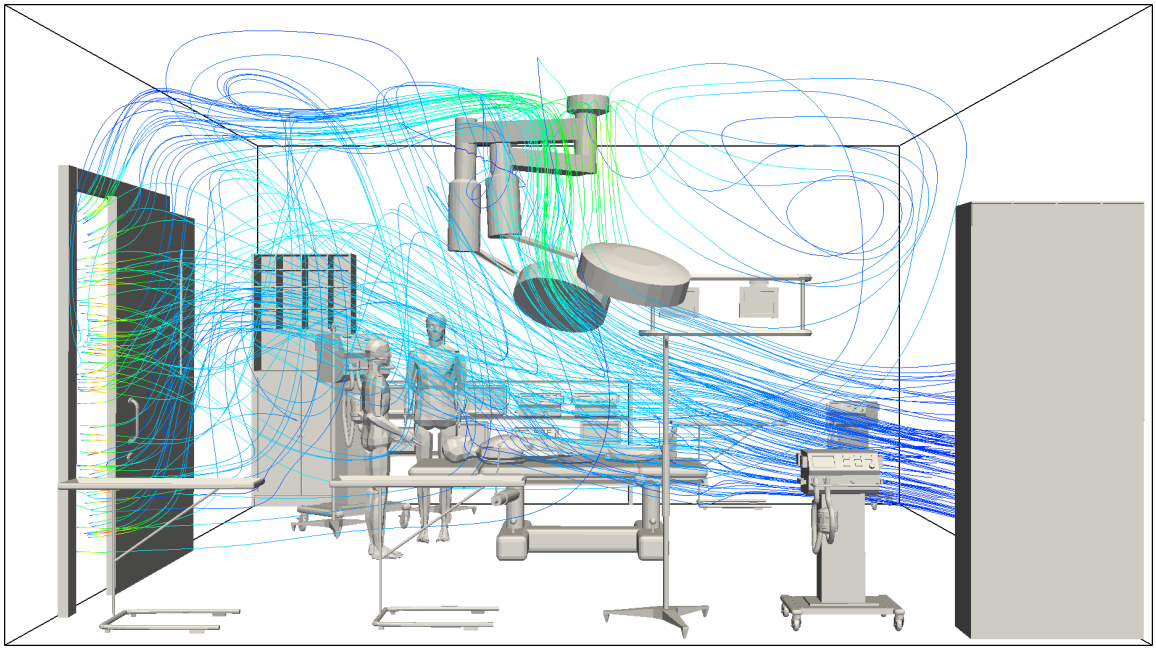


(b) *adapt*4 setup

Figure 7.10.: Temperature distribution for the *adapt*3 and *adapt*4 setup after 31 seconds of elapsed simulation time for $Ra = 10^6$.

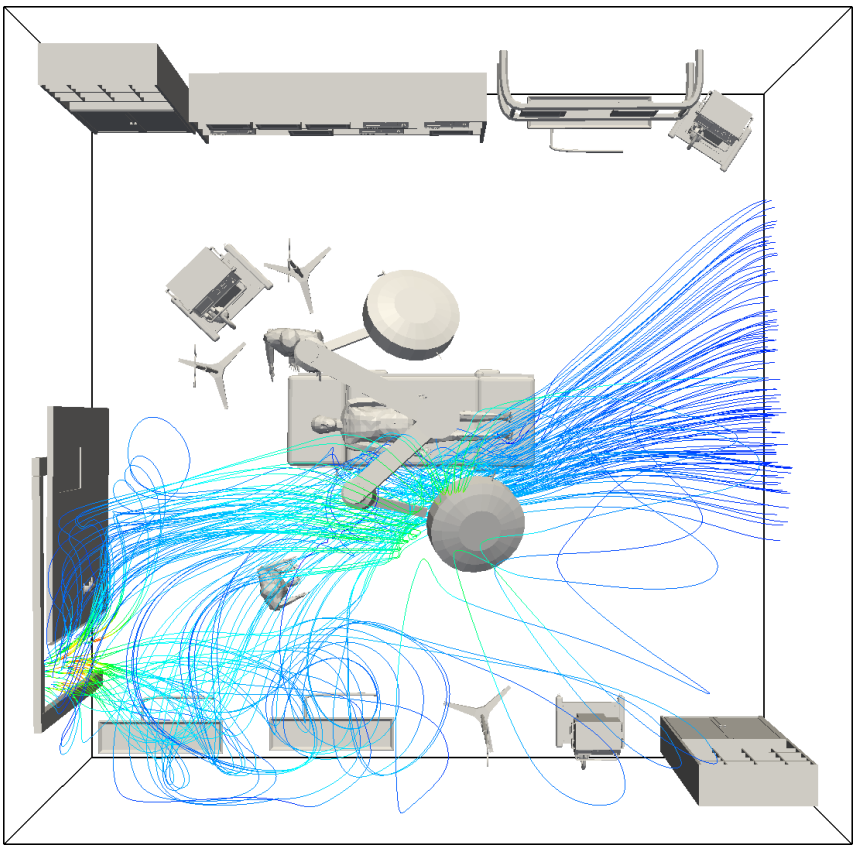


(a) front view

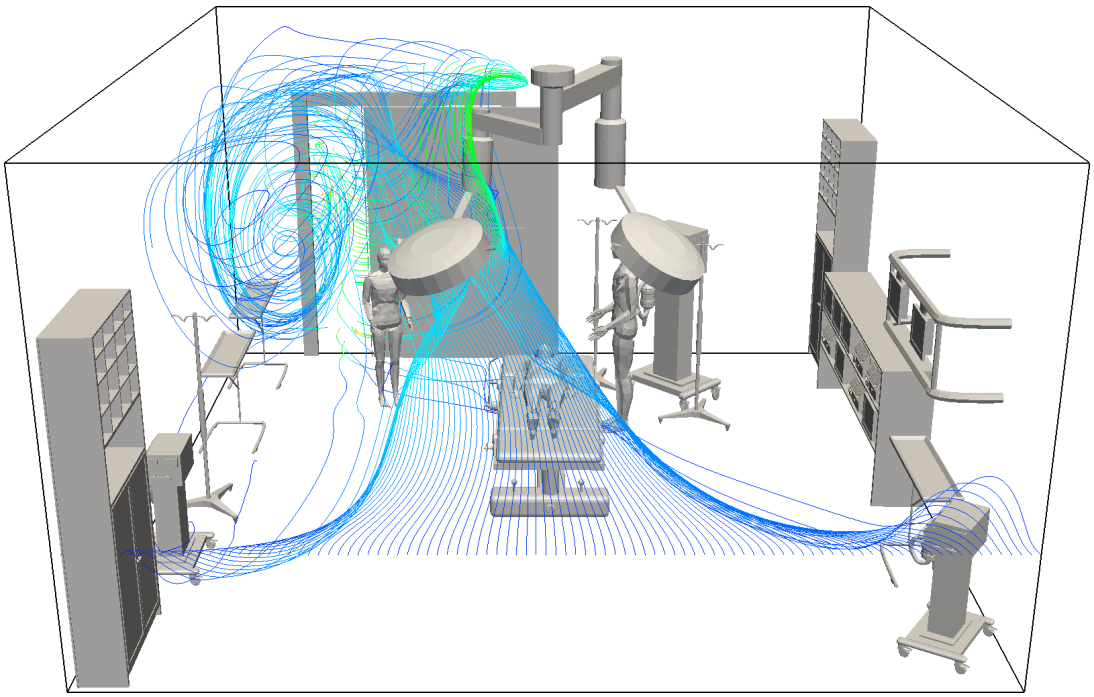


(b) side view

Figure 7.11.: Streamlines using a radial seed point distribution after 100 seconds of elapsed simulation time for $Ra = 10^6$.



(a) top view



(b) front view (line source)

Figure 7.12.: Streamlines using either a radial seed point distribution or a line distribution after 100 seconds of elapsed simulation time for $Ra = 10^6$.

tributed over a sphere with the centre (3.15,3.10,1.75) and a radius of 0.63 m. It can be clearly seen that the two surgical lamps have a huge impact on the fluid flow behaviour in terms of 'convection rolls' similar to the Rayleigh-Bénard convection described in Section 6.5. In Figure 7.12(b) a line source was used for generating seed points. It can be observed that a quite uniform fluid flow behaviour is continuously disturbed by the influence of the lamps and the asymmetrical outlet.

In a second stage, a thermal comfort evaluation as mentioned in the previous section could be performed using the simulated velocities and temperatures from this complex scenario as input values.

7.5. Short Summary

This chapter showed the embedding of the CFD code into a larger software component for simulating an example motivated by an engineering question of thermal comfort assessment. As the data structure can be adapted very easily to the requirements of such an analysis, a segmentation of the obstacles in the room could be performed within a matter of hours, such that different boundary conditions could be applied on the surface of the manikin in a transient way during the run of a CFD simulation. Thermal evaluations according to standards such as [ASH55], [EN15251], or [ISO7730] can be easily used together with the simulation results obtained to generate additional comfort assessment information, which could not be obtained by a zonal simulation alone, due to a lack of specific detailed information.

Furthermore, the sliding-window concept presented in Section 5.2 can be used here for an interactive visualisation and data exploration, as well as an evaluation of thermal comfort assessment using the simulated CFD results.

8. Conclusion and Outlook

Extremely powerful computing systems are gaining ever more in importance in the scientific landscape, creating a wide range of possibilities which were previously beyond imagination in the field of engineering-based fluid flow simulations.

This work has introduced all the steps necessary to create a massive parallel fluid flow code from scratch for solving problems such as thermally coupled indoor air flow simulations, and which is able to run on non-accelerator-based distributed memory clusters or supercomputing systems.

The code currently available supports the solution of an elliptic partial differential equation such as the Poisson equation on a block-structured adaptive l-grid setup, thus forming the basis of a fluid flow computation according to Chorin's projection method applied to the Navier-Stokes equations. The thesis highlighted the excellent scaling properties for up to nearly 80 billion cells on SuperMUC, a supercomputing system currently ranked in the top 10 of the world's most powerful computers. Furthermore, the current code solves and couples the heat transfer equation to the momentum conservation equations by means of the Boussinesq approximation, providing a researcher with a powerful simulation tool for solving indoor air flow scenarios with buoyancy driven fluid flows, for example.

The biggest advantage over similar approaches throughout the research community is the ability of a flexible data structure to distribute, organise, simulate, and visualise data efficiently on one highly adaptive l-grid structure on computing systems ranging from desktops and small clusters to supercomputing systems, while keeping the innermost computational kernel as flexible and exchangeable as possible.

Data exchange mechanisms were devised in order to ensure the consistent synchronisation of the ghost halo to all d-grids, and a multi-grid-like solver approach was proposed which is deeply integrated into the data structure itself and uses the already present ghost cell exchange mechanisms. This concept also enables the back-coupling of user input to a running simulation and thus has de facto a computational steering environment as a byproduct.

Validation results highlight the behaviour of the code for simulations of standard benchmarks such as the driven cavity, a channel flow with and without obstacles, as well as flows driven by natural convection by heated side walls or the well-known Rayleigh-Bénard convection. They are in good agreement given the simplicity of the main simulation kernel which uses an explicit temporal discretisation.

A larger scenario simulating a human manikin in a room showed the possibilities of coupling the CFD simulation to a zonal simulation and a human thermoregulation model according to Fiala. The results are very promising and show good interaction of the multi-scale simulations on the CFD as well as the zonal simulation side.

In comparison to other solutions such as OpenFOAM, the current approach offers very fast and flexible access to a distributed data structure for massive parallel computations, and an easy and straightforward way of integrating a data segmentation in order to identify parts of the geometry, for example parts of the numerical manikin. OpenFOAM, on the other hand, already offers a lot of validated kernels which can be applied without large tests before moving to the interesting problem at hand. Similarly, when compared to all the codes mentioned in the Introduction (i. e. WaLBerla, Virtual Fluids, SciRUN, etc.), a lot of strong points arguing in favour of the approach presented can be identified, but there are also weak points, for which the code was not designed or which were not of primary interest. Thus, the approach presented is very suitable for its area of application and is a valuable extension of the scientific landscape of CFD codes, but could further profit from ideas and concepts integrated in comparable projects.

In conclusion, it can be said that the simulation program with the detailed pipeline which has been created works very well for massive parallel systems, but can still be expanded and/or improved in some points. Possible future developments and improvements are described in the following paragraphs.

Turbulence : The turbulence models were briefly introduced in Chapter 2 but have not yet been implemented in the code, as they were not the main focus of this work. While turbulence would be essential for indoor air flows using realistic air properties, the errors made were nevertheless in an acceptable range for the purpose of this work. It would not be possible to accurately simulate air flows of an urban region without turbulence modelling, however.

Immersed Boundary Conditions : In order to reduce the error in the modelling of boundaries, especially if an object is not aligned along the main axes of the simulation domain, or in the case of non-rectangular obstacles such as the cylinder in the Schäfer-Turek benchmark, immersed boundary conditions should be implemented, as they offer a higher order treatment of boundaries instead of a step-wise approximation of boundaries.

Implicit Time Treatment : The explicit time stepping method has the advantage of a 'cheap' time discretisation, as no system of linear equations has to be solved. On the other hand, depending on the fluid criteria, too large a time step might be unstable and diverge. Hence, small time steps have to be used and, thus, depending on the CFL conditions, might enlarge the time to solution compared to applying an implicit time stepping scheme. Detailed comparisons using both methods should be made, in order to provide 'guidelines' on when to use which method.

Mixed Treatment of Convective and Diffusive Terms : For higher Reynolds or Rayleigh numbers, i. e. for fluids with a low viscosity, oscillations can occur while apply-

ing the central difference discretisation of the convective terms. For these terms, a mixed discretisation using central differences and upwind differences should be applied (donor-cell scheme), in order to stabilise computations for lower viscosities.

Sub-Cycling : Currently, one global time step is applied for all d-grids. As this poses no problem for a uniform d-grid discretisation, an adaptive setting needs to adapt the time step size according to the CFL stability conditions to the finest cell available in the complete domain. Thus, in the worst case, a very local deep refinement is performed, while the rest of the domain is relatively uniform on a coarser level. Nevertheless, the very small time step has to be applied everywhere. This can be obviated by applying a sub-cycling time stepping scheme, meaning that the time step is chosen so as to be smaller on finer d-grids and larger on coarser d-grids, eliminating the need for one global (small) time step. Kazhyken implemented a sub-cycling procedure into the framework proposed during the course of his Master thesis [Kaz13]. Although sub-cycling showed very promising results when applied to the proposed concept, a few unanswered questions remain, such as the coupling of the sub-cycling procedures to the multi-grid-like solver concept, which have to be studied carefully.

Run Time Adaptivity : As soon as a good fluid-based error estimation criterion is defined, run time adaptive refinements can be performed on massive parallel systems using the current data structure. The migration procedures are already available and currently used for the initial ld-grid migration onto the different computing processes. A detailed study has to be performed, analysing how often (i. e. every γ time step) a migration and load balancing should be performed. An optimal setting for γ has to be found, which reduces the time to solution. If the migration is done too often, too much time is spent sending ld-grids back and forth over the network. If the migration and load balancing is not done often enough, however, a poor load balancing will also extend the time to solution.

Accelerators : As mentioned in the Introduction, more and more supercomputers are equipped with accelerators such as GPGPUs. Furthermore, the commodity hardware market is bringing GPGPUs at affordable prices into more and more personal computers and small clusters and thus enabling small engineering companies, for example, to invest in 'high-performance' computers for an affordable price. With this in mind, porting the code presented onto graphic cards or other accelerators such as the Intel Xeon Phi might be very advantageous. The data structure itself is already inherently designed to support and benefit from such a concept. One or more d-grids can be put on one accelerator, and only the ghost halos have to be exchanged at certain times in the main kernel routines. A highly hybrid programming concept can be applied, by performing the inter-node communication with MPI, the intra-node computations with OpenMP[1], and the programming of the graphic card with CUDA[2] for example. Further time should be invested into this point, in order to fully profit from the future trend in supercomputing and to support fast computations on low-budget clusters.

[1] `http://openmp.org`
[2] CUDA (Compute Unified Device Architecture): `https://developer.nvidia.com/cuda-zone`

Interleaving Communication and Computation : In order to reduce the time to solution, one current practice is to interleave communication and computation, thus hiding communication behind computation procedures. If this communication hiding can be done in an efficient way, it will greatly enhance the scalability and the time to solution, as shown for example by [SAT+11] for which they won the ACM Gordon Bell Prize in 2011. Unfortunately, this would destroy the strict separation of communication and computation necessary to have a computational kernel unaffected by parallel communication routines. Thus, the code should/could be developed in two different directions. The version with the strict separation could be used for methodology and kernel development, where experts in their respective fields can work on the integration of a method of higher accuracy, for example. On the other hand, a highly efficient parallel code specifically tuned for high-performance massive parallel systems can be created by using fine granular communication mechanisms in the main kernel, thus providing communication hiding and run time reductions.

Computational Steering Approach : As the sliding window visualisation principle requires a two-way communication between the front-end client on a desktop and the back-end simulation on a cluster or supercomputer in any case, an inherent enhancement would be to integrate a computational steering approach. Until now, the client sends only requests for visualisation purposes, but it could send updates of boundary conditions or geometry modifications as well, due to the fact that all communication operators are already present and, thus, provide a complete computational steering framework coupled to a massive parallel simulation back-end.

All in all, it can be said that a lot of functionalities are already available in the current version, paving the way to future potential in order to create a very powerful, efficient, and fast simulation environment.

Appendix A.

Data Structure Glossary and Definitions

In order to establish a common terminology, all definitions relating to the newly introduced data structure are listed below together with a short explanation.

- *l-grid* : the logical grids or l-grids form the back-bone of the data structure and are organised in a hierarchic, tree-like manner. They contain topological as well as geometrical information about the location of the grids in the structure and are depicted on the right-hand side of Figure 3.10.
- *parent l-grid* : a parent l-grid lg_p of an l-grid lg at depth d is located at depth $d-1$ and 'contains' the aforementioned lg. Each l-grid can only have one parent l-grid.
- *child l-grid* : a child l-grid lg_c of an l-grid lg at depth d is located at depth $d+1$. If an l-grid lg is refined, it must contain $r_x^s \cdot r_y^s \cdot r_z^s$ l-grid children.
- *root l-grid* : the root l-grid is a logical grid which has no parent l-grid, i. e. it is per definition located at depth 0. The root l-grid has per definition $r_x^t \cdot r_y^t \cdot r_z^t$ l-grid children.
- *refined l-grid* : an l-grid is called refined if it has child l-grids.
- *non-refined l-grid* : an l-grid is called non-refined if it has no child l-grids.
- *d-grid* : the data grids or d-grids contain the actual numerical variables which are used for computations such as velocities, pressure, temperature, etc. Numerical stencil computations are performed exclusively on d-grids and they can be compared to the grids introduced in the Sections 3.1 or 3.2. A d-grid itself has no information about its geometrical or topological location, but has a link to one and only one l-grid. Vice-versa, each l-grid has a link to one and only one d-grid. The d-grids can also be called blocks.
- *cell* : a cell is comparable to one finite volume (or control volume) and contains one and only one variable set, i. e. one temperature value, one pressure value, one x-velocity, etc. stored in the centre (for the chosen collocated arrangement). One d-grid contains exactly $s_x \cdot s_y \cdot s_z$ cells (not counting the halo cells).

- *ld-grid* : an ld-grid is an abbreviation for one l-grid and its corresponding linked d-grid without the necessity to mention both the l- and d-grid.
- *domain* : a domain is a set of ld-grids. In the case of a serial computation, there exists one domain containing all l-grids and all d-grids. In a parallel computation, each process has one domain containing a set of l-grids and their corresponding d-grids. An ld-grid can only belong to one domain, i. e. it has a fixed affiliation to one domain and, thus, to one process.

Appendix B.

Computing and Visualisation Platforms

In order to execute simulations and measurements, different computing platforms were used. Detailed information about these platforms are listed in order to be comparable.

B.1. Desktop Environments

B.1.1. Ivy-Bridge Desktop

The desktop system contains an Intel Ivy-Bridge quad-core Intel Core i7-3770 CPU at 3.40 GHz and 16 GB RAM. This is the first system on which parts of the code were developed and tested. Using hyper-threading technology, two threads per process can be spawned. During normal operational conditions, this is quite effective, but for scientific purposes such as presented in this work, performance measurements showed that hyper-threading is counter-productive and it was subsequently deactivated.

B.2. Cluster and Supercomputer Environments

B.2.1. CiE Sandstorm Cluster

`http://www.cie.bgu.tum.de`

The Sandstorm cluster is a small 4 node cluster installed in August 2012 at the Chair for Computation in Engineering at TUM. Each node contains two Intel Xeon E5-2690 CPUs running at 2.9 GHz and having 192 GB RAM. The network interconnect is based on a one-to-one 1 Gb Ethernet connection from every node to every node, and a 1 Gb connection to an external switch. The sustained Linpack performance[1] of the system is 1.3 TFLOPS. Hyper-threading capabilities have been deactivated.

[1] `http://www.top500.org/project/linpack`

B.2.2. TUM MAC-Cluster

`http://www.mac.tum.de`

The Munich Centre of Advanced Computing at TUM operates a heterogeneous research cluster hosted at the Leibniz Supercomputing Centre LRZ. It is composed of 36 nodes having dual socket Intel SandyBridge-EP Xeon E5-2670 CPUs with 128 GB RAM, where 4 nodes have NVIDIA M2090 GPUs and 4 nodes have AMD FirePro W8000 GPUs. Furthermore, there are 2 Intel Westmere-EX Xeon E7-4830 nodes with 512 GB RAM and 19 AMD Bulldozer Opteron 6274 nodes with 256 GB RAM. All nodes are connected through InfiniBand[2] connections of various speeds (either QDR or FDR).

B.2.3. SuperMUC

`http://www.lrz.de/services/compute/supermuc`

The Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities operates one of Germany's three supercomputer systems, called SuperMUC, which went into operation in June 2012, when it was ranked in the Top 500 list as number four in the world and number one in Europe for computing performance[3].

It consists of 155,656 processor cores in 9,400 compute nodes distributed over 19 islands and using FDR10 InfiniBand as interconnect, organised as a 4:1 pruned tree for the inter-island interconnects and as a non-blocking tree for the intra-island interconnects. The peak performance is over 3 PFLOPS and the sustained performance is measured as 2.897 PFLOPS.

SuperMUC uses a new, revolutionary form of warm water cooling developed by IBM. Active components like processors and memory are directly cooled with water that can have an inlet temperature of up to 40 degrees Celsius. The 'High Temperature Liquid Cooling' together with very innovative system software promise to cut the energy consumption of the system. In addition, all LRZ buildings will be heated re-using this energy[4].

B.2.4. UVT Blue Gene/P

`http://hpc.uvt.ro/infrastructure/bluegenep`

The Universitatea de Vest din Timişoara (UVT) in Romania operates a fully loaded single IBM Blue Gene/P rack that has 1,024 compute cards, each comprising a quad-core PowerPC-450 at 850 MHz, 4 GB RAM per CPU, and a 3D torus network interconnect between the computing cards. The IO nodes are connected by a 10 Gb Ethernet connection. It can

[2]InfiniBand Trade Association (IBTA) : `http://www.infinibandta.org`
[3]Source: `http://www.top500.org/lists/2012/06`
[4]Source: `http://www.lrz.de/services/compute/supermuc/systemdescription`

offer 11.7 TFLOPS sustained performance and is currently the fastest supercomputer in Romania.

B.2.5. KAUST Blue Gene/P 'Shaheen'

`http://www.hpc.kaust.edu.sa`

Shaheen is a 16 rack IBM Blue Gene/P supercomputer named after the Arabic peregrine falcon, which is a very fast bird that reaches speeds of up to 112 km/h in level flight. During its spectacular hunting swoop from heights of over 1,000 m, the Shaheen may reach speeds of 320 km/h as it dives on its prey[5]. Each node is equipped with an 850 MHz PowerPC-450 quad-core processor and 4 GB DDR memory. In aggregate, Shaheen has 65,536 compute cores and 64 TB of memory. Shaheen provides a three-dimensional point-to-point Blue Gene/P torus network for general-purpose IPC. Each torus link can transmit up to 425 MBps in each direction, for a total of 5.1 GBps bidirectional bandwidth per node.

Shaheen was initially ranked at position 14 in the Top 500 list[6] during the International Supercomputing Conference in June 2009. Falling back to position 214 in November 2013, Shaheen still remains the most powerful supercomputer in the Middle East with 190.9 TFLOPS sustained performance.

B.3. Visualisation Environments

B.3.1. KAUST AESOP

The AESOP system is a 40-tile display wall installed at the visualisation core lab at KAUST in Saudi Arabia. It is composed of 46" NEC ultra-narrow bezel panels, each having a resolution of $1{,}360 \times 768$ pixels that enable large-scale, near-seamless HD images without any use of projection[7]. It is powered by a computing cluster of 10 servers each having four graphic cards, plus a dedicated front-end system.

[5]Source: `http://www.hpc.kaust.edu.sa/about/KAUST_Corelab_Supercomputing.pdf`
[6]Source: `http://www.top500.org/list/2009/06`
[7]Source: `http://www.vis.kaust.edu.sa/`

Bibliography

[ASH55] ASHRAE Standard 55. *Thermal Environmental Conditions for Human Occupancy*. American Society of Heating, Refrigerating and Air-Conditioning Engineers, 2004.

[Bad13] M. Bader. *Space-Filling Curves – An Introduction with Applications in Scientific Computing*. Texts in Computational Science and Engineering. Springer-Verlag, 2013.

[Bat00] G. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge Mathematical Library. Cambridge University Press, 2000.

[Bej13] A. Bejan. *Convection Heat Transfer*. John Wiley & Sons, 4th rev. edition, 2013.

[Bén01] H. Bénard. *Les tourbillons cellulaires dans une nappe liquide propageant de la chaleur par convection, en régime permanent*. Gauthier-Villars, 1901.

[BFR80] J. Bardina, J. H. Ferziger, and W. C. Reynolds. Improved subgrid-scale models for large-eddy simulation. In *Fluid Dynamics and Co-located Conferences*. American Institute of Aeronautics and Astronautics, July 14-16, 1980.

[BHM00] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd rev. edition, 2000.

[BHW02] H.-J. Bungartz, H.-J. Herrmann, and B. I. Wohlmuth. Simulierte Welten – die Zukunft im Rechner. *Wechselwirkungen – Jahrbuch aus Lehre und Forschung der Universität Stuttgart*, pages 34–50, 2002.

[BL05] M. R. Benioff and E. D. Lazowska. Report to the President. Computational Science: Ensuring America's Competitiveness. *President's Information Technology Advisory Committee (PITAC)*, June 2005.

[BL11] A. Brandt and O. Livne. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 2011.

[Bra77] A. Brandt. Multi-Level Adaptive Solutions to Boundary-Value Problems. *Mathematics of Computation*, 31(138):333–390, April 1977.

[BSMM08] I. Bronstein, K. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Harri Deutsch, 2008.

[CFL28] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100(1):32–74, 1928.

[Cho67] A. J. Chorin. Numerical solution of the Navier-Stokes equations. *Mathematics of Computation*, 22:745–762, 1967.

[CJ97] C.-J. Chen and S.-Y. Jaw. *Fundamentals Of Turbulence Modelling*. Combustion: An International Series. Taylor & Francis, 1997.

[CL93] B. Cabral and L. C. Leedom. Imaging vector fields using Line Integral Convolution. In *Proc. of the 20th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH*, pages 263–270, Anaheim, CA, USA, August 02-06, 1993. ACM.

[Cla01] J. A. Clarke. *Energy simulation in building design*. Butterworth-Heinemann, 2nd edition, 2001.

[CM94] H. Choi and P. Moin. Effects of the computational time step on numerical solutions of turbulent flow. *Journal of Computational Physics*, 113(1):1–4, July 1994.

[CSG99] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. The Morgan Kaufmann Series in Computer Architecture and Design Series. Morgan Kaufmann Publishers, 1999.

[Cyb89] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7(2):279–301, 1989.

[dDB02] R. J. de Dear and G. S. Brager. Thermal comfort in naturally ventilated buildings: revisions to ASHRAE standard 55. *Energy and Buildings*, 34(6):549–561, 2002. Special Issue on Thermal Comfort Standards.

[DFF+03] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White. *Sourcebook of Parallel Computing*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2003.

[ISO7730] DIN EN ISO 7730. *Ergonomics of the thermal environment – Analytical determination and interpretation of thermal comfort using calculation of the PMV and PPD indices and local thermal comfort criteria*. Beuth Verlag, May 2006.

[EN15251] DIN EN 15251. *Indoor environmental input parameters for design and assessment of energy performance of buildings addressing indoor air quality, thermal environment, lighting and acoustics*. Beuth Verlag, August 2007.

[DSYX06] H. Ding, C. Shu, K. Yeo, and D. Xu. Numerical computation of three-dimensional incompressible viscous flows in the primitive variable form by local multiquadric differential quadrature method. *Computer Methods in Applied Mechanics and Engineering*, 195(7-8):516–533, 2006.

[DVD83] G. De Vahl Davis. Natural convection of air in a square cavity: A benchmark numerical solution. *International Journal for Numerical Methods in Fluids*, 3(3):249–264, 1983.

[ES95] J. G. M. Eggels and J. A. Somers. Numerical simulation of free convective flow using the Lattice-Boltzmann scheme. *International Journal of Heat and Fluid Flow*, 16(5):357–364, 1995.

[Fan70] P. Fanger. *Thermal comfort: Analysis and applications in environmental engineering*. Danish Technical Press, 1970.

[Fer98] J. H. Ferziger. *Numerical Methods for Engineering Applications*. Wiley-Interscience Publication. Wiley, 1998.

[FGD+09] C. Feichtinger, J. Götz, S. Donath, K. Iglberger, and U. Rüde. WaLBerla: Exploiting massively parallel systems for Lattice-Boltzmann simulations. In *Parallel Computing*, pages 241–260. Springer-Verlag, 2009.

[FK95] U. Frisch and A. N. Kolmogorov. *Turbulence: The Legacy of A. N. Kolmogorov*. Cambridge University Press, 1995.

[FLS99] D. Fiala, K. J. Lomas, and M. Stohrer. A computer model of human thermoregulation for a wide range of environmental conditions: the passive system. *Journal of Applied Physiology*, 87(5):1957–1972, November 1999.

[FLS01] D. Fiala, K. J. Lomas, and M. Stohrer. Computer prediction of human thermoregulatory and temperature responses to a wide range of environmental conditions. *International Journal of Biometeorology*, 45(3):143–159, September 2001.

[FM10] J. Frisch and R.-P. Mundani. Multiskalen-Strömungssimulation eines Kraftwerkskomplexes auf Höchstleistungsrechnern. In *Proc. of the 22nd Forum Bauinformatik*, pages 59–66, Berlin, Germany, September 29-October 01, 2010.

[FMR11a] J. Frisch, R.-P. Mundani, and E. Rank. Adaptive data structure management for grid based simulations in engineering applications. In *Proc. of the 8th International Conference on Scientific Computing*, Las Vegas, NV, USA, July 18-21, 2011.

[FMR11b] J. Frisch, R.-P. Mundani, and E. Rank. Communication schemes of a parallel fluid solver for multi-scale environmental simulations. In *Proc. of the 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 391–397, Timişoara, Romania, September 26-29, 2011. IEEE Computer Society.

[FMR12] J. Frisch, R.-P. Mundani, and E. Rank. Resolving neighbourhood relations in a parallel fluid dynamic solver. In *Proc. of the 11th International Symposium on Parallel and Distributed Computing*, pages 267–273, München, Germany, June 25-29, 2012. IEEE Computer Society.

[FMR13a] J. Frisch, R.-P. Mundani, and E. Rank. Adaptive distributed data structure management for parallel CFD applications. In *Proc. of the 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 513–520, Timişoara, Romania, September 23-26, 2013. IEEE Computer Society.

[FMR13b] J. Frisch, R.-P. Mundani, and E. Rank. Parallel multi-grid like solver for the pressure Poisson equation in fluid flow applications. In *Proc. of the IADIS International Conference – Applied Computing*, pages 139–146, Fort Worth, TX, USA, October 23-25, 2013.

[FMR14] J. Frisch, R.-P. Mundani, and E. Rank. Adaptive multi-grid methods for parallel CFD applications. *Scalable Computing: Practice and Experience*, 15(1):33–48, April 2014.

[FP02] J. H. Ferziger and M. Perić. *Computational Methods for Fluid Dynamics*. Springer-Verlag, 3rd rev. edition, 2002.

[Fri08] J. Frisch. Parametrisches Modell zur interaktiven Simulation menschlicher Thermoregulation. Diploma thesis, Technische Universität München, Germany, August 2008.

[FvDFH96] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice in C*. Addison-Wesley, 2nd edition, 1996.

[FVOMY00] E. Fadlun, R. Verzicco, P. Orlandi, and J. Mohd-Yusof. Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations. *Journal of Computational Physics*, 161(1):35–60, 2000.

[FvT09] J. Frisch and C. van Treeck. Virtual Climate Chamber – Entwicklung einer interaktiven Simulationsumgebung für thermische Komfortanalysen. In *Proc. of the 21st Forum Bauinformatik*, pages 67–78, Karlsruhe, Germany, September 23-25, 2009.

[Gag73] A. Gagge. Rational temp. indices of man's thermal env. and their use with a 2-node model of his temp. reg. *Fed. Proc.*, 32:1572–1582, 1973.

[GDN98] M. Griebel, T. Dornsheifer, and T. Neunhoeffer. *Numerical Simulation in Fluid Dynamics: A Practical Introduction*. Monographs on Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics, 1998.

[GGS82] U. Ghia, K. N. Ghia, and C. T. Shin. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics*, 48(3):387–411, 1982.

[GK10] L. Grinberg and G. Karniadakis. A new domain decomposition method with overlapping patches for ultrascale simulations: Application to biological flows. *Journal of Computational Physics*, 229(15):5541–5563, 2010.

[GKSR14] B. Gmeiner, H. Köstler, M. Stürmer, and U. Rüde. Parallel multigrid on hierarchical hybrid grids: a performance study on current high performance computing clusters. *Concurrency and Computation: Practice and Experience*, 26(1):217–240, 2014.

[GKT+06] S. Geller, M. Krafczyk, J. Tölke, S. Turek, and J. Hron. Benchmark computations based on Lattice-Boltzmann, finite element and finite volume methods for laminar flows. *Computers & Fluids*, 35(8-9):888–897, 2006.

[Hac10] W. Hackbusch. *Multi-Grid Methods and Applications*. Volume 4 of Springer Series in Computational Mathematics. Springer-Verlag, 2010.

[Had05] H. Hadžić. *Development and Application of Finite Volume Method for the Computation of Flows Around Moving Bodies on Unstructured, Overlapping Grids*. PhD thesis, Technische Universität Hamburg-Harburg, Germany, December 2005.

[Hir07] C. Hirsch. *Numerical Computation of Internal and External Flows, Volume 1*. Butterworth-Heinemann, 2nd edition, 2007.

[HNR75] C. W. Hirt, B. D. Nichols, and N. C. Romero. SOLA: a numerical solution algorithm for transient fluid flows. Technical Report LA-5852, Los Alamos National Laboratory, Los Alamos, NM, USA, January 1975.

[HW65] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8(12):2182–2189, 1965.

[Jan94] R. J. A. Janssen. *Instabilities in natural-convection flows in cavities*. PhD thesis, Technische Hogeschool, Delft, Netherlands, May 1994.

[Kaz13] K. Kazhyken. Implementing an Adaptive Time Stepping Scheme by Sub-Cycling in a CFD Code. Master's thesis, Technische Universität München, Germany, September 2013.

[KHT87] H. C. Ku, R. S. Hirsh, and T. D. Taylor. A pseudospectral method for solution of the three-dimensional incompressible Navier-Stokes equations. *Journal of Computational Physics*, 70(2):439–462, 1987.

[KKC01] J. Kim, D. Kim, and H. Choi. An immersed-boundary finite-volume method for simulations of flow in complex geometries. *Journal of Computational Physics*, 171(1):132–150, 2001.

[KM85] J. Kim and P. Moin. Application of a fractional-step method to incompressible Navier-Stokes equations. *Journal of Computational Physics*, 59(2):308–323, 1985.

[Kol42] A. N. Kolmogorov. Equations of Turbulent Motion of an Incompressible Fluid. *Izvestiya Akademii Nauk SSSR, Seriya Fizicheskaya*, 6:56–58, 1942.

[KTI+08] Y. Kurazumi, T. Tsuchikawa, J. Ishii, K. Fukagawa, Y. Yamato, and N. Matsubara. Radiative and convective heat transfer coefficients of the human body in natural convection. *Building and Environment*, 43(12):2142–2153, 2008.

[Leb04] H. L. Lebesgue. *Leçons sur l'intégration et la recherche des fonctions primitives*. Gauthier-Villars, Paris, France, 1904.

[Leo75] A. Leonard. Energy cascade in large-eddy simulations of turbulent fluid flows. In *Turbulent Diffusion in Environmental Pollution Proceedings of a Symposium held at Charlottesville*, volume 18, Part A of *Advances in Geophysics*, pages 237–248. Elsevier, 1975.

[LL11] J. Lienhard IV and J. Lienhard V. *A Heat Transfer Textbook*. Dover Civil and Mechanical Engineering Series. Dover Publications, 4th edition, 2011.

[LQ91] P. Le Quéré. Accurate solutions to the square thermally driven cavity at high Rayleigh number. *Computers & Fluids*, 20(1):29–41, 1991.

[LS72] B. E. Launder and D. B. Spalding. *Mathematical Models of Turbulence*. Academic Press, London, 1972.

[Meh10] M. Mehl. *A Combination of Efficient Numerical and Computer Science Methods for the Simulation of Fluid-Dynamics Applications*. Habilitation thesis, Technische Universität München, Germany, June 2010.

[Mei11] A. Meister. *Numerik linearer Gleichungssysteme : eine Einführung in moderne Verfahren*. Vieweg + Teubner, 4th edition, 2011.

[MFR80] O. J. McMillan, J. H. Ferziger, and R. S. Rogallo. Tests of subgrid-scale models in strained turbulence. In *Fluid Dynamics and Co-located Conferences*. American Institute of Aeronautics and Astronautics, July 1980.

[MFR13] R.-P. Mundani, J. Frisch, and E. Rank. Towards interactive HPC: Sliding Window Data Transfer. In *Proc. of the 3rd International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering (ParEng)*, Pécs, Hungary, March 25-27, 2013. Civil-Comp Press.

[Mih51] S. G. Mihlin. On the algorithm of Schwarz. *Doklady Akademii Nauk SSSR, n. Ser.*, 77:569–571, 1951.

[Moo65] G. E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117, April 1965.

[Mor66] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd., Ottawa, ON, Canada, 1966.

[MST10] A. McAdams, E. Sifakis, and J. Teran. A parallel multigrid Poisson solver for fluids simulation on large grids. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, Madrid, Spain, July 02-04, 2010.

[Mun05] R.-P. Mundani. *Hierarchische Geometriemodelle zur Einbettung verteilter Simulationsaufgaben*. PhD thesis, Universität Stuttgart, Germany, December 2005.

[Mun13] R.-P. Mundani. *Interaction in High-Performance Computing Scenarios*. Habilitation thesis, Technische Universität München, Germany, July 2013.

[MW94] M. Manhart and H. Wengle. Large-eddy simulation of turbulent boundary layer flow over a hemisphere. In *1st ERCOFTAC Workshop on 'Direct and Large-eddy Simulation'*, Surrey, UK, March 27-30, 1994.

[PDR07] J. Parvizian, A. Düster, and E. Rank. Finite cell method. *Computational Mechanics*, 41(1):121–133, 2007.

[Pfa12] M. Pfaffinger. *Interaktive Strömungssimulation auf Hochleistungsrechnern unter Anwendung der Lattice-Boltzmann Methode*. PhD thesis, Technische Universität München, Germany, April 2012.

[Phi84] T. N. Phillips. Natural convection in an enclosed cavity. *Journal of Computational Physics*, 54(3):365–381, 1984.

[PJ95] S. G. Parker and C. R. Johnson. SCIRun: A scientific programming environment for computational steering. In *Proc. of the 1995 ACM/IEEE Conference on Supercomputing*, San Diego, CA, USA, December 04-08, 1995.

[Pra45] L. Prandtl. *Über ein neues Formelsystem für die ausgebildete Turbulenz*. Nachrichten der Akademie der Wissenschaften in Göttingen, Mathematisch-Physikalische Klasse. Vandenhoeck & Ruprecht, Göttingen, 1945.

[PS40] A. Pellew and R. V. Southwell. On maintained convective motion in a fluid heated from below. *Proc. of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 176(966):312–343, 1940.

[PT83] R. Peyret and T. Taylor. *Computational Methods for Fluid Flow*. Springer Series in Computational Physics. Springer-Verlag, 1983.

[PWS07] S. Paulke, S. Winter, and T. Schneider. Some considerations on global and local thermal comfort based on Fiala's thermal manikin in THESEUS-FE. In *Proc. of the 6th EUROSIM Congress on Modelling and Simulation*, Ljubljana, Slovenia, September 09-13, 2007.

[Rey95] O. Reynolds. On the dynamical theory of incompressible viscous fluids and the determination of the criterion. *Philosophical Transactions of the Royal Society of London*, vol. 186:123–164, 1895.

[RS86] W. Rodi and G. Scheuerer. Scrutinizing the k-ε turbulence model under adverse pressure gradient conditions. *Journal of Fluids Engineering*, 108(2):174–179, June 1986.

[Sag06] P. Sagaut. *Large Eddy Simulation for Incompressible Flows*. Springer-Verlag, 3rd rev. edition, 2006.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Boston, MA, USA, 1990.

[SAT+11] T. Shimokawabe, T. Aoki, T. Takaki, T. Endo, A. Yamanaka, N. Maruyama, A. Nukada, and S. Matsuoka. Peta-scale Phase-field Simulation for Dendritic Solidification on the TSUBAME 2.0 Supercomputer. In *Proc. of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 11)*, Seattle, WA, USA, November 12-18, 2011.

[Sch70] H. A. Schwarz. Ueber einen Grenzübergang durch alternirendes Verfahren. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 15:272–286, May 1870.

[SK11] H. R. Schwarz and N. Köckler. *Numerische Mathematik*. Vieweg + Teubner, 8th rev. edition, 2011.

[Sma63] J. Smagorinsky. General circulation experiments with the primitive equations. *Monthly Weather Review*, 91(3):99–164, March 1963.

[ST96] M. Schäfer and S. Turek. Benchmark computations of laminar flow around a cylinder. In *Flow Simulation with High-Performance Computers II*, volume 52 of *Notes on Numerical Fluid Mechanics*, pages 547–566. Vieweg, January 1996.

[Sto71] J. A. J. Stolwijk. A mathematical model of physiological temperature regulation in man. Contractor report NASA CR-1855, National Aeronautics and Space Administration, Washington, D.C., USA, 1971.

[Sto91] J. A. J. Stolwijk. Sick-building syndrome. *Environmental Health Perspectives*, 95:99–100, 1991.

[SvTB+11] S. Stratbücker, C. van Treeck, S. R. Bolineni, D. Wölki, and A. Holm. A co-simulation framework for scale-adaptive coupling between heterogeneous computational codes. In *Proc. of the 12th ROOMVENT Conference*, Trondheim, Norway, June 19-22, 2011.

[TF03] Y.-H. Tseng and J. H. Ferziger. A ghost-cell immersed boundary method for flow in complex geometry. *Journal of Computational Physics*, 192(2):593–623, 2003.

[TL72] H. Tennekes and J. L. Lumley. *A First Course in Turbulence*. MIT Press, 1972.

[TM94] M. F. Tome and S. McKee. GENSMAC: A computational marker and cell method for free surface flows in general domains. *Journal of Computational Physics*, 110(1):171–186, 1994.

[TOS01] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001.

[VDI6020] VDI 6020 Part 1. *Requirements on methods of calculation to thermal and energy simulation of buildings and plants*. Verein Deutscher Ingenieure, May 2001.

[vK30] T. von Kármán. Mechanische Ähnlichkeit und Turbulenz. In *Proc. of the 3rd. International Congress of Applied Mechanics*, Part 1, pages 85–105, Stockholm, Sweden, 1930.

[vT04] C. van Treeck. *Gebäudemodell-basierte Simulation von Raumluftströmungen.* PhD thesis, Technische Universität München, Germany, June 2004.

[vT10] C. van Treeck. *Introduction to Building Performance Modeling and Simulation.* Habilitation thesis, Technische Universität München, Germany, February 2010.

[vTFER09] C. van Treeck, J. Frisch, M. Egger, and E. Rank. Model-adaptive analysis of indoor thermal comfort. In *Proc. of the 11th International IBPSA Conference on Building Simulation*, pages 1374–1381, Glasgow, Scotland, July 27-30, 2009.

[vTFP+09] C. van Treeck, J. Frisch, M. Pfaffinger, E. Rank, S. Paulke, I. Schweinfurth, R. Schwab, R. Hellwig, and A. Holm. Integrated thermal comfort analysis using a parametric manikin model for interactive real-time simulation. *Journal of Building Performance Simulation*, 2(4):233–250, 2009.

[vTM13] C. van Treeck and M. Mitterhofer. Temperaturfeldberechnung aus einer Particle Image Velocimetry (PIV)-Messung einer natürlichen Auftriebsströmung. *Bauphysik*, 35(2):77–85, 2013.

[Wen08] P. Wenisch. *Computational Steering of CFD Simulations on Teraflop-Supercomputers.* PhD thesis, Technische Universität München, Germany, February 2008.

[Whi91] F. M. White. *Viscous Fluid Flow.* McGraw-Hill Education, 2nd rev. edition, 1991.

[Wil94] D. C. Wilcox. *Turbulence Modeling for CFD.* DCW Industries, Inc., 2nd rev. edition, 1994.

[WSJ+12] P. C. Wong, H.-W. Shen, C. R. Johnson, C. Chen, and R. B. Ross. The top 10 challenges in extreme-scale visual analytics. *IEEE Computer Graphics and Applications*, 32(4):63, 2012.

[YO86] V. Yakhot and S. A. Orszag. Renormalization group analysis of turbulence. I. Basic theory. *Journal of Scientific Computing*, 1(1):3–51, 1986.

[ZTZ05] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals.* Elsevier Science, 6th edition, 2005.