Implementation and Performance Analyses of a Highly Efficient Algorithm for Pressure-Velocity Coupling

Masterarbeit | Fabian Gabel, Studienbereich CE
Betreuer: Dipl.-Ing. U. Falk | Prof. Dr. rer. nat. M. Schäfer

TECHNISCHE UNIVERSITÄT DARMSTADT

fnb Fachgebiet Numerische Berechnungsverfahren im Maschinenbau

Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

Date: Signature:

Abstract

One crucial decision in designing algorithms in order to solve the incompressible Navier-Stokes equations numerically is how to treat the inherent coupling between the dependent variables, velocity and pressure. Due to advances regarding processor and memory hardware technology during the past decades, fully coupled solution algorithms have become attractive in the field of CFD (*Computational Fluid Dynamics*). These algorithms denominate methods which simultaneously solve a system of partial differential equations for all involved variables. The objective of the present thesis is to implement and analyze a pressure-based, fully coupled solution algorithm based on a finite-volume discretization of the Navier-Stokes equations including additional coupling to the temperature equation. The method uses a cell-centered, co-located variable arrangement and is designed to handle non-orthogonal, three-dimensional, locally refined, block-structured grids with hanging nodes. The implementation has been parallelized using the PETSc (*Portable Extensible Toolkit for Scientific Computation*) framework. After the verification using a manufactured solution, the single processor and parallel performance of the developed application are analyzed on a parallel benchmark to assess strong and weak scalability. A benchmark involving a duct flow with complex geometry and a temperature-driven cavity benchmark for buoyancy-driven flows are performed. The performance analysis always consists of a comparison with an implementation of a segregated solution algorithm, which was developed using the same environment. Furthermore, three correctors addressing non-orthogonality of numerical grids are analyzed with respect to their effect on the convergence of the coupled solution algorithm. The conducted test shows that the *over-relaxed approach* is the most robust for different degrees of non-orthogonality of the grid. A simple technique to achieve ideal load balancing for the linear equation solvers is presented. The effects of this method on solver performance for an unbalanced data partitioning is assessed, showing that parallel calculations benefit from this load balancing technique. In all tests, the fully coupled algorithm outperformed the segregated algorithm, provided that the number of involved unknowns was high enough. The solution of the associated linear systems was carried out using black-box solvers from the PETSc framework. It is assumed that significant improvements regarding the parallel scalability of the implemented algorithm may be achieved by special algebraic multigrid methods as preconditioner to the PETSc Krylov subspace solvers.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Due to the advancements in hardware technology and solution algorithms, the involvement of numerical methods in sciences and industrial practice has increased significantly during the past decades, having an enormous impact on academic research and processes in industrial development. Besides theoretical or experimental approaches, numerical methods nowadays form an integral part in fostering a better understanding of physical phenomena and in modeling the behavior of technical systems. Numerical methods are used for the simulation of fluid flow problems, a branch of numerics also known as *CFD* (*Computational Fluid Dynamics*). CFD continuously reaches the limits of current simulation environments, creating new challenges to further improvements of the deployed hardware and software. CFD methods not only have to respond to a higher level of complexity, which may include complex geometries or multiphysics, but they also are expected to generate results after shorter periods of simulation. In order to overcome these challenges, research is done into the implementation of novel and more efficient algorithms; apart from that, CFD applications often rely on the efficient usage of high-performance computer clusters.

In the field of CFD, the incompressible Navier-Stokes equations represent the commonly used mathematical model for low Mach number fluid flows. Among other properties, this model couples two variables, velocity and pressure. Thus, algorithms that tend to solve for incompressible flow fields have to address the so-called pressure-velocity coupling. A standard approach is to resolve this coupling through segregated solution methods. By these methods, the algebraic equations for each of the dependent variables are solved sequentially, using approximate values for the other involved variables. One common representative of this kind of algorithms is the SIMPLE-algorithm, being widely used in scientific codes or industrial solvers due to its small memory requirements and the efficient iterative solvers which exist for the resulting linear systems. Reference [**?**] reviews the evolution of this kind of algorithms.

The use of segregated solution algorithms entails significant deficiencies. In order to stabilize the iteration for the coupling process, the obtained solutions have to be under-relaxed. In doing so, the degree of under-relaxation also depends on the problem to be solved. Not only does this degrade the solver's performance, but, making things even more difficult, there also do not exist strict rules for the best selection of the amount of under-relaxation. Furthermore, algorithms like SIMPLE lack scalability with the mesh size. Resolving the pressure-velocity coupling defines the flow solver's performance. Consequently, new coupling algorithms, more robust or even independent of parameters, and, at the same time, resource efficient in their application, have to be developed. [**?**] offers a helpful analysis of the pressure-velocity coupling.

1.1 State of Research and Knowledge

Considering the solution approach, pressure-based solution algorithms for incompressible Navier-Stokes equations can be divided into segregated solution methods and semi-direct or coupled solution methods. Segregated solution methods, as for example the well-known SIMPLE-algorithm [**?**], have been actively used during the past decades. One common property of segregated solution algorithms is the two-step solution approach, involving a predictor step and a corrector step. In the first step, a velocity field is calculated based on an estimation of the pressure field, in the second step a pressure equation or a pressure-correction equation is solved. After this, the velocities are corrected. Many variants of segregated solution methods have been developed over the time, aiming to increase the robustness and efficiency of the SIMPLE-algorithm. The other chiefly used ones, besides the SIMPLE-algorithm, are SIMPLER (SIMPLE Revised) [**?**], SIMPLEC (SIMPLE Consistent) [**?**], and PISO (Pressure Implicit with Splitting of Operator) [**?**], which modify the SIMPLE-algorithm. The main advantages of segregated solution algorithms are low memory requirements and the straightforward extension to the solution of additional partial differential equations. [**?**] gives an overview of segregated solution algorithms.

The SIMPLER-algorithm addresses the inherent problem of the need to under-relax the variables throughout the solution process. Based on the observation that the pressure-correction based velocity-corrections are satisfactory with respect to the improvement of the solution, the pressure-corrections have to be under-relaxed before they are applied to the pressure itself, as pressure-corrections generally tend to be too high. In order to remove the need to under-relax the pressure-correction, the pressure-correction from the SIMPLE algorithm is only used for velocity corrections, whereas a separate pressure equation based on the corrected velocities is derived. The SIMPLEC-algorithm proposes another remedy to the under-relaxation problematic by using a consistent simplification in the derivation of the pressure-correction equation.

The SIMPLER-algorithm is found to reduce the number of iterations for convergence efficiently while increasing the computational effort and having higher memory requirements. The SIMPLEC-algorithm also leads to a smaller, compared

to the SIMPLER-algorithm, decrease in the number of iterations for convergence, while there is no notable increase in the computational effort.

The PISO-algorithm, as does the SIMPLE-algorithm, uses a pressure-correction step to correct the velocities. The pressure-correction step is then, different to the SIMPLE-algorithm, followed by further correction steps to compensate for the simplifications made in the derivation of the pressure-correction equation. Similar to the other modifications, the PISO-algorithm leads to a reduced iteration count for convergence and lacks the necessity to under-relax the pressure-correction. One known drawback of the PISO-algorithm is the higher computational effort and the higher memory requirements.

An alternative solution approach to the problem of numerically solving the Navier-Stokes equations is by means of coupled solvers which simultaneously solve the momentum and mass balance equations. Algorithms of this type, using finite-volume methods, have been presented in [**?**, **?**, **?**, **?**, **?**, **?**, **?**]. [**?**] shows that the principal advantages of coupled solution approaches compensate for the deficiencies of segregated solution algorithms. All mentioned references report that, on the one hand, the use of coupled solution methods reduces the computational time, while, on the other hand, the solution algorithm's robustness increases. According to [**?**] no under-relaxation is necessary at all to obtain a robust implementation. [**?**] reported insensitivity towards the relaxation allowing faster convergence compared to frequently used segregated solution algorithms. In all mentioned references the two-step, predictor-corrector solution procedure was reduced to one step. An equation for the pressure was deduced using a momentum interpolation scheme [**?**] in all cited references but [**?**].

[**?**] extended the linear system for velocities and pressure by an additional decoupled block accounting for temperature. So far, only [**?**, **?**] have used coupled solution processes for flow problems implicitly involving temperature transport. The mentioned references extended the scope of a coupled solution process for the two-dimensional Navier-Stokes equations to include a temperature equation. They investigated the effect of different methods on implicit temperature-to-velocity/pressure coupling via a Newton-Raphson linearization on the solver's performance, showing that the use of coupled methods significantly reduces the number of iterations until convergence. The objectives of the present thesis are to implement a fully-coupled solution algorithm for the three-dimensional Navier-Stokes equations and the temperature equation on co-located grids and to analyze their single and multi-processor performance. The methods to achieve temperature coupling are similar to the ones presented in [**?**]. In this reference, the treatment of the interpolation of the volumes in the pressure weighted interpolation method is different from the implementation used in the present thesis. The implementation in [**?**] relies on a geometric partition of the existing volumes other than using a linear interpolation. Another difference results from the used interpolation technique for pressure gradients at faces, for which [**?**] uses local coordinate systems and a wider computational stencil.

1.2 Structure of the Thesis

In chapter 2, the thesis introduces the mathematical and physical fundamentals of flow problems presenting the main frame of partial differential equations that have to be solved numerically. Each section outlines a derivation from the continuum mechanical point of view, and shows common simplifications, comprising not only the Navier-Stokes equations for incompressible fluids but also the Boussinesq approximation, which is widely used for incompressible flows when experiencing buoyancy forces.

Chapter 3 outlines the fundamentals of finite-volume methods in general. At first, the necessary terminology for numerical grids is introduced. After that, the main approximation principles for the discretization of integral or differential operators are presented, placing particular emphasis on the modifications necessary for discretizing them on non-orthogonal grids. In the end, this chapter outlines the principal method of linearization for the Navier-Stokes equations used throughout the work.

Chapter 4 focuses on segregated solution methods because many ideas of the commonly used SIMPLE-algorithm are also relevant for the implementation of coupled solution methods. The chapter starts by discretizing the mass balance coming directly from the continuum mechanical introduction of chapter 2, outlining its drawbacks on their applicability to co-located variable arrangements. Section 4.2 introduces a remedy to the *checker-boarding* effect surging from an unmodified discretization of the mass balance by proposing a modification of the commonly used Rhie-Chow momentum interpolation scheme [**?**]. This modification yields results which do not depend on under-relaxation. This fact is important for later comparison studies with the coupled solution algorithm. In section 4.3, the familiar SIMPLE-algorithm is derived, using the previously demonstrated interpolation scheme as a basis for the construction of a pressure-correction equation. In sections 4.4, 4.5, and 4.6, the pressure equation and the two remaining sets of equations, i.e. the three momentum balances and the temperature equation, are discretized. Each section concludes with a complete list of the resulting discretized equations and calculations of the coefficients as implemented in the developed solver framework. Section 4.7 is dedicated to the implementation of boundary conditions, presenting the necessary modifications to the discretized equations for each boundary condition, inclusive of the treatment of block boundaries. Since in most cases the boundary conditions lead to a singular system for the pressure-correction, section 4.8 describes methods to constrain the pressure

in incompressible flows. This chapter concludes by presenting the non-zero structure of the resulting assembled linear system, taking into account the special treatment of block boundaries with hanging nodes.

Chapter 5 describes the implemented coupled solution algorithm. The structure of this chapter resembles the structure of chapter 4 since both solution approaches share an enormous part of their components. For this reason, this thesis puts particular emphasis on the differences between both algorithms. Section 5.1 reconsiders the pressure-velocity coupling that accounts for the implicit consideration of the pressure in the momentum balances. A major difference from segregated solution methods is then presented in section 5.2: the coupling to the temperature equation. Here, different methods, varying in their relation of implicit to explicit coupling of velocities, pressure, and temperature, are deduced. After that, the necessary modifications for boundary conditions are presented. Section 5.4 shows the final form of the discretized equations and the resulting structure of the assembled linear systems, placing special emphasis on the effects of the different methods leading to implicit coupling of the momentum balances to the temperature equation and vice versa.

Chapter 6 introduces the developed *CAFFA* (*Computer Aided Fluid Flow Analysis*) framework, having its theoretical basis in the preceding chapters. This framework is linked to the PETSc library, which is a sophisticated toolkit for the parallelisation of solver applications. PETSc also provides a set of efficient preconditioners and Krylov subspace solvers for the solution of linear systems. Sections 6.2 and 6.3 then introduce the other building blocks of the framework, beginning with a grid generator for arbitrarily refined block-structured grids and a preprocessor program and concluding with implementation details of the CAFFA solvers. These details comprise the used message passing model, the control of convergence and the data decomposition in the context of parallel computation.

Chapter 7 presents the results of a conducted convergence study to verify the CAFFA framework. After briefly introducing the theory of the verification of scientific codes through manufactured solutions, the author proposes a manufactured solution for the particular kind of Navier-Stokes equations that use the Boussinesq approximation. Then, the results of a grid convergence study are presented to prove the order of accuracy of the developed solvers within the CAFFA framework. The chapter concludes with the outcome of a short study on the effect which the under-relaxation factors of the velocities have on the final result of calculations.

Chapter **??** presents the results of the performance tests conducted on the high-performance cluster *HHLR* (*Hessischer Hochleistungsrechner*) of TU Darmstadt. Having introduced the used hardware and software configuration and the relevant performance metrics, section **??** presents program runtime measurements and the calculated application speedup. Furthermore, this section shows the results of a study conducted on the weak scalability of the implemented fully coupled solution algorithm. Section **??** and section **??** present performance analyses regarding the efficiency of coupled and segregated solution algorithms for flow problems considering complex geometries and buoyant forces. Section **??** deals with the effect of different types of non-orthogonal correctors on the convergence of the coupled solution algorithm. This chapter concludes with section **??** proposing a simple load balancing technique based on automatic matrix partitioning.

The last chapter **??** summarizes the results of the conducted research work, giving an outlook on further studies.

2 Fundamentals of Continuum Physics for Thermo-Hydrodynamical Problems

This chapter covers the set of fundamental equations for thermo-hydrodynamical problems which the numerical solution techniques of the following chapters are aiming to solve. Furthermore, this chapter introduces the notation regarding the physical quantities to be used throughout this thesis. The following paragraphs are based on [**?**, **?**, **?**, **?**]. For a thorough derivation of the matter to be presented the reader may consult the mentioned sources. Since the present thesis focuses on the application of finite-volume methods, the focus lays on stating the integral forms of the relevant conservation laws. However, the process of deriving the final set of equations requires the use of differential formulations of the stated laws. Einstein's convention for taking sums over repeated indices simplifies certain expressions. The remainder of this thesis uses non-moving inertial frames in a Cartesian coordinate system with the coordinates $x_i, i = 1, ..., 3$. This approach is also known as *Eulerian approach*.

2.1 Conservation of Mass – Continuity Equation

The conservation law of mass, also known as the continuity equation, embraces the physical concept that, neglecting relativistic effects and nuclear reactions, mass cannot be created or destroyed. Using the notion of a mathematical control volume to denote a constant domain of integration, one can state the integral mass balance of a control volume V with control surface S with surface unit normal vector $\mathbf{n} = (n_i)_{i=1,...,3}$ using Gauss's theorem of integration as follows

$$\iiint_V \frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(\rho u_i)\,\mathrm{d}V = \iiint_V \frac{\partial \rho}{\partial t}\mathrm{d}V + \iint_S \rho u_i n_i \mathrm{d}S = 0.$$

Here, ρ denotes the material density, t denotes the independent variable of time and $\mathbf{u} = (u_i)_{i=1,...,3}$ is the velocity vector field. Since this equation remains valid for arbitrary control volumes, the equality has to hold for the integrands as well. In this sense, the differential form of the conservation law of mass can be formulated as

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(\rho u_i) = 0. \tag{2.1}$$

2.2 Conservation of Momentum – Cauchy-Equations

The conservation law of momentum, also known as Newton's Second Law, axiomatically demands the balance of the temporal change of momentum and the sum of all attacking forces on a body. These forces comprise body forces and surface forces. Let $\mathbf{k} = (k_i)_{i=1,...,3}$ denote a mass-specific force and $\mathbf{t} = (t_i)_{i=1,...,3}$ the stress vector. A preliminary formula of the integral momentum balance in the direction of x_i reads as follows

$$\iiint_V \frac{\partial}{\partial t}(\rho u_i)\,\mathrm{d}V + \iint_S \rho u_i \left(u_j n_j\right)\mathrm{d}S = \iiint_V \rho k_i \mathrm{d}V + \iint_S t_i \mathrm{d}S. \tag{2.2}$$

In general, the stress vector $\mathbf{t}$ is a function not only of the location $\mathbf{x} = (x_i)_{i=1,...,3}$ and of the time t but also of the surface unit normal vector $\mathbf{n}$. A central simplification can be introduced, namely Cauchy's stress theorem, which states that the stress vector is the image of the unit normal vector under a linear mapping $\mathbf{T}$. With respect to the Cartesian canonical basis $(\mathbf{e}_i)_{i=1,...,3}$, the mapping $\mathbf{T}$ is represented by the coefficient matrix $\left(\tau_{ji}\right)_{i,j=1,...,3}$, and Cauchy's stress theorem reads

$$\mathbf{t}(\mathbf{x}, t, \mathbf{n}) = \mathbf{T}(\mathbf{x}, t, \mathbf{n}) = \left(\tau_{ji} n_j\right)_{i=1,...,3}.$$

Assuming the validity of Cauchy's stress theorem, one can derive Cauchy's first law of motion, which in differential form reads as follows

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}\left(\rho u_i u_j\right) = \rho k_i + \frac{\partial \tau_{ji}}{\partial x_j}, \tag{2.3}$$

representing the starting point for the modeling of fluid mechanical problems. Cauchy's first law of motion does not make any assumptions regarding material properties. For that reason, the set of equations (2.1,2.3) is not closed, i.e. there does not exist an independent equation for each of the dependent variables.

2.3 Closing the System of Equations – Newtonian Fluids

As a result of Cauchy's theorem, the stress vector $\mathbf{t}$ can be calculated once the nine components τ_{ji} of the coefficient matrix are known. As shown in [**?**, **?**], by formulating the conservation law of angular momentum the coefficient matrix is symmetric

$$\tau_{ji} = \tau_{ij}. \tag{2.4}$$

Hence, the number of unknown coefficients may be reduced to six unknown components. In a further step, it is assumed that the coefficient matrix can be decomposed into fluid-static and fluid-dynamic contributions,

$$\tau_{ij} = -p\delta_{ij} + \sigma_{ij},$$

where p is the thermodynamic pressure, δ_{ij} is the *Kronecker*-Delta and σ_{ij} is the so-called *deviatoric stress tensor*.

The present study models viscous fluids using a linear relation between the components of the deviatoric stress tensor and the symmetric part of the transpose of the Jacobian matrix of the velocity field $\left(S_{ij}\right)_{i,j=1,\dots,3}$,

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right).$$

If one now imposes material-isotropy and the mentioned stress-symmetry (2.4) restriction, it can be shown [**?**] that the constitutive equation for the deviatoric stress tensor reads

$$\sigma_{ij} = 2\mu S_{ij} + \lambda S_{mm}\delta_{ij}.$$

Here λ and μ denominate scalars which depend on the local thermodynamical state. Taking everything into account, (2.3) can be reformulated as the differential conservation law of momentum for Newtonian fluids, better known as the *Navier-Stokes equations* in differential form:

$$\frac{\partial}{\partial t}\left(\rho u_i\right) + \frac{\partial}{\partial x_j}\left(\rho u_i u_j\right) = \rho k_i - \frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right) + \frac{\partial}{\partial x_i}\left(\lambda\frac{\partial u_m}{\partial x_m}\right) \tag{2.5}$$

2.4 Conservation of Scalar Quantities

The modeling of the transport of scalar or vector quantities by a flow field $\mathbf{u}$, also known as convection, is necessary if the fluid mechanical problem to be analyzed includes, for example, heat transfer. Other scenarios that involve the necessity to model scalar transport surge, when turbulent flows are to be modelled by two-equation models like the k-ε-model [**?**].

Since this thesis focuses on the transport of the scalar temperature T, this section introduces the conservation law of energy in differential form, formulated in terms of the temperature,

$$\frac{\partial}{\partial t}\left(\rho T\right) + \frac{\partial}{\partial x_j}\left(\rho u_j - \kappa\frac{\partial T}{\partial x_j}\right) = q_T.$$

Here κ denotes the thermal diffusivity of the modelled material and q_T is a scalar field representing sources and sinks of heat throughout the domain of the problem. This equation is also known as the temperature equation.

2.5 Necessary Simplification of Equations

The purpose of this section is to introduce and justify further common simplifications of the previously presented set of constitutive equations.

2.5.1 Incompressible Flows and Hydrostatic Pressure

A common simplification when modeling low Mach number flows ($Ma < 0.3$), is the assumption of *incompressibility*, or the assumption of an *isochoric* flow. Assuming homogeneous density ρ in space and time, a restriction that will be partially loosened in the following section, the continuity equation in differential form (2.1) can be simplified to

$$\frac{\partial u_i}{\partial x_i} = 0.$$

In other words: In order for a velocity vector field $\mathbf{u}$ to be valid for incompressible flow, it has to be free of divergence, in other terms *solenoidal* [**?**, **?**].

Furthermore, assuming the dynamic viscosity μ to be constant, which can be suitable in the case of isothermal flow or small temperature differences within the flow, the Navier-Stokes equations in differential form can be reduced to

$$\rho \frac{\partial}{\partial t}(u_i) + \rho \frac{\partial}{\partial x_j}(u_i u_j) = \rho k_i - \frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right) \tag{2.6a}$$

$$= \rho k_i - \frac{\partial p}{\partial x_i} + \mu \frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j}\right). \tag{2.6b}$$

Here, *Schwartz*'s lemma to interchange the order of differentiation, has been used. Another common step to further simplify the set of equations is the assumption of a volume-specific force $\rho\mathbf{k}$ that can be modelled by a potential in such a way that it can be represented as the gradient of a scalar field $\Phi_\mathbf{k}$ as

$$-\rho k_i = \frac{\partial \Phi_\mathbf{k}}{\partial x_i}.$$

In the case of this thesis, this assumption is valid, since the mass-specific force is the mass-specific gravitational force $\mathbf{g} = (g_i)_{i=1,\dots,3}$, and the density is assumed to be constant, so that the potential can be modeled as

$$\Phi_g = -\rho g_j x_j.$$

This term can be interpreted as the hydrostatic pressure p_{hyd} and can be added to the thermodynamical pressure p to simplify calculations

$$\begin{aligned}
\rho g_i - \frac{\partial p}{\partial x_i} &= \frac{\partial}{\partial x_i}(\rho g_j x_j) - \frac{\partial p}{\partial x_i} \\
&= \frac{\partial}{\partial x_i}(\rho g_j x_j) - \frac{\partial}{\partial x_i}(\hat{p} + p_{hyd}) \\
&= -\frac{\partial \hat{p}}{\partial x_i}.
\end{aligned}$$

Since to incompressible fluids only pressure differences matter, this has no effect on the solution. After finishing the calculations, p_{hyd} can be calculated and added to the resulting pressure $\hat{p}$.

2.5.2 Variation of Fluid Properties – The Boussinesq Approximation

Modeling of an incompressible flow with heat transfer has to take into account that fluid properties, as density, change with varying temperature. If the variation of temperature is small, one can still assume constant density to maintain the structure of the advection and diffusion terms in (2.5), and only consider changes of the density in the gravitational term. Assuming linear variation of density with respect to temperature, this approximation is called *Boussinesq* approximation [**?**]. This approximation furthermore consists in assuming that all other fluid properties are constant and viscous dissipation can be neglected. In this case, the Navier-Stokes equations are formulated using a reference density ρ_0 at the reference temperature T_0 and the following temperature-dependent density ρ

$$\rho(T) = \rho_0(1 - \beta(T - T_0)).$$

Here, β denotes the coefficient of thermal expansion. Using the Boussinesq approximation, the incompressible Navier-Stokes equations in differential form can be formulated as

$$\begin{aligned}\rho_0 \frac{\partial u_i}{\partial t} + \rho_0 \frac{\partial}{\partial x_j}\left(u_i u_j\right) &= \rho_0 g_i + (\rho - \rho_0) g_i - \frac{\partial p}{\partial x_i} + \mu \frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \\ &= \frac{\partial}{\partial x_i}\left(\rho_0 g_j x_j\right) + (\rho - \rho_0) g_i - \frac{\partial p}{\partial x_i} + \mu \frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \\ &= -\frac{\partial \hat{p}}{\partial x_i} + (\rho - \rho_0) g_i + \mu \frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \\ &= -\frac{\partial \hat{p}}{\partial x_i} - \rho_0 \beta \left(T - T_0\right) g_i + \mu \frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\end{aligned}$$

using $\rho \mathbf{g}$ as the mass-specific force.

2.6 Final Form of the Set of Equations

In the previous sections, different simplifications have been introduced which are used throughout the thesis. The final form of the set of equations to be used is thereby presented. As a further simplification, the modified pressure $\hat{p}$ will be treated as p, and since the utilization of the Boussinesq approximation replaces the variable ρ by a linear function of the temperature T, the reference density ρ_0 for the remainder of this thesis is referred to as ρ. The following set of equations takes incompressibility into account

$$\frac{\partial u_i}{\partial x_i} = 0. \tag{2.7a}$$

$$\rho \frac{\partial u_i}{\partial t} + \rho \frac{\partial}{\partial x_j}\left(u_i u_j\right) = -\frac{\partial p}{\partial x_i} - \rho \beta \left(T - T_0\right) g_i + \mu \frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{2.7b}$$

$$\frac{\partial}{\partial t}\left(\rho T\right) + \frac{\partial}{\partial x_j}\left(\rho u_j T - \kappa \frac{\partial T}{\partial x_j}\right) = q_T. \tag{2.7c}$$

3 Finite-Volume Methods for Incompressible Flows – Theoretical Basics

This chapter deals with the fundamentals of the numerical solution via a finite-volume method for the formerly presented set of partial differential equations (2.7). The focus of this chapter is providing an overview of the methods used in the present thesis. The information contained in this chapter is based on [**?**, **?**, **?**, **?**]. The overview starts by mentioning the relevant grid types and the discretization techniques. On the basis of integral formulations of the equations to be solved, the contained integrals and differential operators have to be discretized. This chapter furthermore presents different approaches to the handling of corrections for cases of increased non-orthogonality which are causing degradation of grid quality.

The finite-volume method aims at providing particular linear algebraic equations to determine an approximate solution of a partial differential equation. However, since the Navier-Stokes equations are nonlinear, an intermediate step is necessary before linear systems can be derived and solvers for linear equation systems can be applied: The discrete equations have to be linearized. This condition leads to the need for an iteration process, the *Picard iteration*, which this thesis will also briefly explain.

3.1 Numerical Grid

This section provides a brief overview of the grid structure relevant to the present thesis. One central idea behind finite-volume methods is to solve partial differential equations by integrating them over the specified continuous problem domain and dividing this domain into a finite number of subdomains, the so-called *control volumes*. The result of this finite partition of a contiguous problem domain is called numerical grid. Every grid consists of a finite number of control volumes representing the boundaries of a discrete domain of integration. Depending on whether the numerical solution of an equation is to be calculated on the boundary vertices or in the center of a grid cell, the variable arrangement is denoted to be vertex-oriented or cell center-oriented. The remainder of the present thesis assumes that the variables reside in the cell centers of the respective control volumes.

Another option to arrange variables specifically for algorithms that solve the Navier-Stokes equations is the so-called *staggering*. This method stores the values of different variables at different grid locations. This storage facilitates the interpolation of the respective values. As the methods employed in the present thesis are intended to be applicable to complex geometries, the cell-centered non-staggered approach offers more flexibility. Figure 3.1 shows the different variable arrangement options on a two-dimensional grid.

Regarding the treatment of domain boundaries and the ordering of the cells within the problem domain, different types of numerical grids can be distinguished. The present thesis makes use of so-called boundary-fitted, block-structured grids with hexahedron cells. A structured grid is characterized by a constant amount of grid cells in each coordinate direction. The high regularity of structured grids benefits the computational efficiency of the algorithms to be used on this type of grid. In addition to that, a block-structured grid consists of different grid blocks, of which every single one, considered individually, is structured. Yet, considering the grid as a whole, the grid is unstructured. Figure 3.2 shows an example of a block-structured grid with distinguishable grid blocks. The use of block-structured grids is motivated by the need to increase the adaptivity of structured grids while maintaining high computational efficiency. Furthermore, block-structuring embraces the concept of domain decomposition which facilitates the implementation of parallel algorithms for the decomposed computational domain. Boundary-fitted grids represent domain boundaries by means of the geometry of the control volumes residing on those boundaries. This approach may lead to skewed cells and affects the accuracy of the results obtained in these boundary cells. Using local grid refinement techniques to refine the grid in regions of geometrically complex boundaries can alleviate these effects.

Inside a structured grid block, cells with the shape of hexahedrons are used. In addition to the geometric boundaries of each control volume, a numerical grid also provides a mapping that assigns a set of indices of neighboring control volumes $NB(P) := \{W, S, B, T, N, E\}$ to each control volume with index P. The indices of the neighboring control volumes are named after the geographic directions. Figure 3.3 shows a single grid cell with its direct neighbors. The faces $\{S_w, S_s, S_b, S_t, S_n, S_e\}$ of each hexahedral control volume represent the mentioned geometric boundaries.

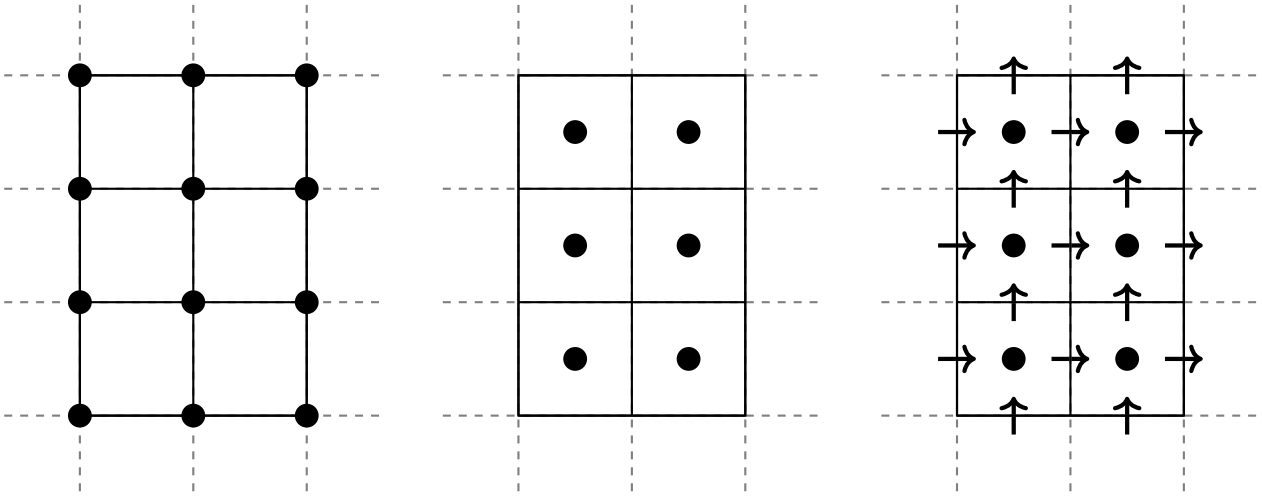


Figure 3.1: Vertex centered, cell-centered and staggered variable arrangement. For the staggered arrangement, the arrows denote the velocity component and their location on the grid. The centered dot denotes the location of the pressure and further scalar quantities

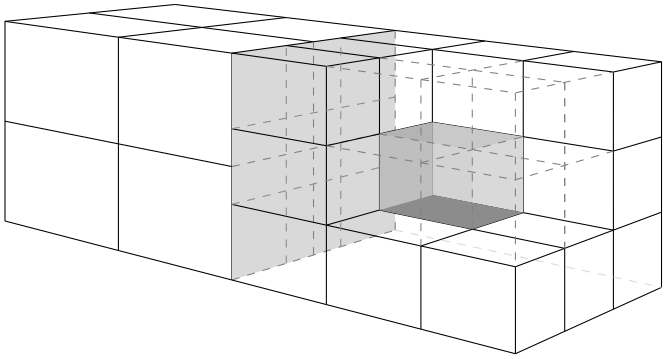


Figure 3.2: Block-structured grid consisting of one $2 \times 2 \times 2$ block and one $3 \times 3 \times 3$ block

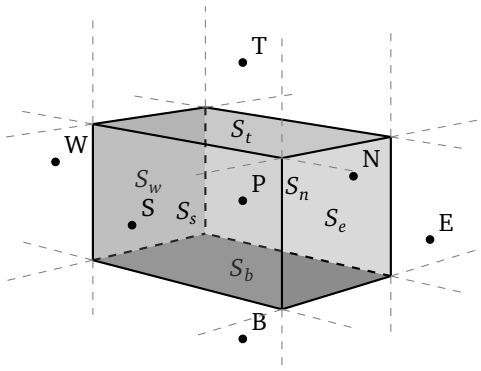


Figure 3.3: Notation used for one control volume, its boundary faces, and its neighbors

3.2 Approximation of Integrals and Derivatives

In the course of transforming a partial differential equation into a system of linear algebraic equations, integrals and derivatives have to be approximated. The simplest method for approximating an integral is by using the *midpoint rule*. This rule is similar to the mean value theorem of integration, which states that there exists a point $\xi \in V$ for a Riemann integrable function ϕ such that $\phi(\xi)\int_V \mathrm{d}V = \int_V \phi(x)\mathrm{d}V$. For the midpoint rule, ξ is taken to be the center of mass of V. If the integration domain V is indeed a volume, fortunately the calculation of $\phi(\xi)$ with $\xi := \left(\int_V x_i \mathrm{d}V / \int_V \mathrm{d}V\right)_{i=1,\ldots,3}$ presents no difficulties since, due to the collocated variable arrangement, the value of ϕ is stored in the cell center which corresponds to the location ξ. However, if the domain of integration is a surface, a preceding interpolation step is necessary.

Furthermore, in order to transform a partial differential equation into a linear algebraic equation, it is necessary to discretize the differential operators of the equations. For numerical reasons, two different discretization techniques are used in this thesis. A common task is discretizing expressions of the form

$$(\nabla\phi)_e \cdot \mathbf{n}_e,$$

where $(\nabla\phi)_e$ is the gradient of ϕ on the boundary face S_e. One method is to interpret this expression directly as a directional derivative and approximate it with a central difference

$$(\nabla\phi)_e \cdot \mathbf{n}_e \approx \frac{\phi_P - \phi_E}{||\mathbf{x}_P - \mathbf{x}_E||_2}. \tag{3.1}$$

Another method would be to first calculate the cell center gradients $(\nabla\phi)_P$ and $(\nabla\phi)_E$ and interpolate them linearly before calculating the projection onto $\mathbf{n}_e$

$$(\nabla\phi)_e \cdot \mathbf{n}_e \approx \left[\gamma_e (\nabla\phi)_P + (1-\gamma_e)(\nabla\phi)_E\right] \cdot \mathbf{n}_e, \tag{3.2}$$

where $\gamma_e := ||\mathbf{x}_P - \mathbf{x}_e||_2 / ||\mathbf{x}_P - \mathbf{x}_E||_2$ is a geometric interpolation factor. For calculating the cell center gradients, a method based on Gauss's theorem of integration and the midpoint rule for volume integration is employed

$$(\nabla\phi)_{i,P} = \frac{\iiint_V \left(\frac{\partial\phi}{\partial x_i}\right) \mathrm{d}V}{\Delta V} \approx \frac{\sum_f \phi_f n_i S_f}{\Delta V}. \tag{3.3}$$

3.3 Treatment of the Non-Orthogonality of Grid Cells

Unfortunately, real applications involve complex geometries, a fact which in turn affects the grid's orthogonality when using boundary fitted grids. On non-orthogonal meshes, the directional derivative in the direction of the face's unit normal vector $\mathbf{n}_e$ can no longer be approximated as in (3.1). On the other hand, the exclusive usage of (3.3) is not desirable due to the bigger truncation error following from this approximation [**?**]. Hence, a compromise is made and the surface vector $\mathbf{S}_e := S_e \mathbf{n}_e$ is decomposed as

$$\mathbf{S}_e = \mathbf{\Delta} + \mathbf{k}, \tag{3.4}$$

where $\mathbf{\Delta}$ is parallel to the vector $\mathbf{d}_e := (\mathbf{x}_E - \mathbf{x}_P)$ that directly connects the center $\mathbf{x}_P$ of the control volume P to the center $\mathbf{x}_E$ of its neighbor E. This vector controls the *orthogonal* contribution to the directional derivative. Vector $\mathbf{k}$ controls the influence of the *non-orthogonal* contribution. In the next paragraphs, the three main decompositions of the surface vector $\mathbf{S}_e$ will be presented by stating the respective expression for $\mathbf{\Delta}$. The resulting vector $\mathbf{k}$ can be calculated by using (3.4). One important characteristic common to all the presented approaches is that the non-orthogonal contribution vanishes when an orthogonal grid is used. For reasons of graphical representation, the decompositions are chosen to be represented two-dimensionally. Figure 3.4 gives a geometrical interpretation of the three approaches. The last section deals with the integration of one generic approach into the discretization process.

The results of a study on the effect of different correctors on the convergence behavior of a solver for partial differential equations can be found in section **??**.

3.3.1 Minimum Correction Approach

[**?**] proposes the *minimum correction approach*. Even though [**?**] references [**?**], [**?**] uses a different approach, which will be presented in the next paragraph. This method is designed to keep the non-orthogonal contribution minimal by always choosing $\mathbf{k}$ to be orthogonal to $\mathbf{\Delta}$, resulting in

$$\mathbf{\Delta} = (\mathbf{d} \cdot \mathbf{S}_e) \frac{\mathbf{d}}{\mathbf{d} \cdot \mathbf{d}}.$$

It is important to note that the influence of the orthogonal contribution decreases with increasing non-orthogonality of the grid.

3.3.2 Orthogonal Correction Approach

[**?**] presents *orthogonal correction approach* for decomposing the surface normal vector. This approach is also implemented in the developed solvers. It uses a simple projection which is independent of the non-orthogonality of the grid. As a result, the orthogonal contribution $||\mathbf{\Delta}||_2 = ||\mathbf{S}_e||_2$ and is thus modeled as

$$\mathbf{\Delta} = ||\mathbf{S}_e||_2 \frac{\mathbf{d}}{||\mathbf{d}||_2}.$$

3.3.3 Over-Relaxed Approach

[?, ?] use the over-relaxed approach which is characterized by an increasing influence of the orthogonal contribution with increasing grid non-orthogonality, as opposed to the minimum correction approach. The orthogonal contribution is calculated as

$$\mathbf{\Delta} = (\mathbf{S}_e \cdot \mathbf{S}_e) \frac{\mathbf{d}}{\mathbf{d} \cdot \mathbf{S}_e}.$$

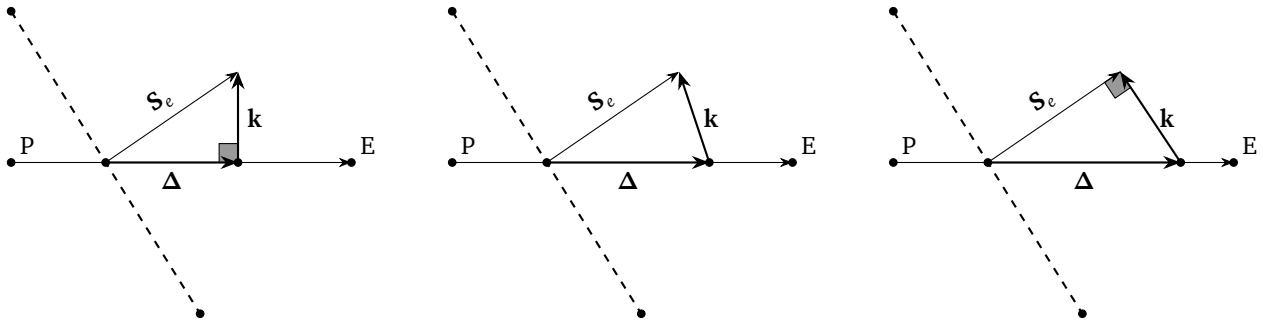


Figure 3.4: Minimum correction, orthogonal correction and over-relaxed approach

3.3.4 Deferred Non-Orthogonal Correction

In order to reduce the computational stencil that would be necessary to handle the non-orthogonal correction implicitly, the correction will be treated explicitly, using a deferred correction which guarantees that in the case of a fully converged solution, only the face normal derivative is taken into account. Generally, the discretization using a non-orthogonal correction would yield

$$(\nabla\phi)_e \cdot \mathbf{S}_e \approx (\nabla\phi)_e \cdot \mathbf{\Delta} + (\nabla\phi)_e \cdot \mathbf{k}.$$

Here, the first term can be approximated using a central differencing scheme for the directional derivative, and the second can be approximated by interpolating cell center gradients. Making use of the fact that this method is applied within a solution algorithm for a nonlinear system of partial differential equations, a deferred correction can be implemented. This correction ensures a smaller error from the non-orthogonality. In the case of the previously mentioned discretization techniques for partial derivatives, a possible deferred correction approach reads

$$(\nabla\phi)_e \cdot \mathbf{S}_e \approx ||\mathbf{\Delta}||_2 \frac{\phi_P - \phi_E}{||\mathbf{x}_P - \mathbf{x}_E||_2} - (\nabla\phi)_e^{(n-1)} \cdot (\mathbf{\Delta} - \mathbf{S}_e). \tag{3.5}$$

Equation (3.5 shows that the use of a deferred correction, in conjunction with the requirement that the non-orthogonal correction vanishes on orthogonal grid, introduces an inconsistent discretization of $(\nabla\phi)_e \cdot \mathbf{\Delta}$.

3.4 Numerical Solution of Nonlinear Systems – Picard Iteration

In the process of solving nonlinear systems, typically two levels of iterations are distinguished: the *inner* iterations which account for the iterations needed to solve a given system of linear equations, and the *outer* iterations which deal with a possible nonlinearity or coupling of the equations. As will be shown in section 4.5.1, outer iterations are also necessary if one chooses the deferred correction approach in order to blend higher and lower order discretization schemes, or, as section 3.3 showed, in order to apply non-orthogonal correctors.

The system of partial differential equations (2.7) is nonlinear due to the terms $\rho u_i u_j$ which are often denoted as the convective terms of the Navier-Stokes equations. It is a standard approach to linearize those terms according to the *Picard-Iteration*. For the convective terms of the Navier-Stokes equations or the temperature equation, this means to guess the mass flux ρu_j and approximate as

$$\rho u_j u_i \approx \left(\rho u_j\right)^o u_i.$$

The same procedure applies to the convective terms of the temperature equation $\rho u_j T$. The advantages of this linearization technique are the low memory requirements and the small computational effort for one iteration. Because of that,

this method is commonly implemented in solution programs for flow problems that use a segregated solution algorithm similar to the one utilized in this thesis. However, compared to Newton-like methods, this approach needs more iterations to converge. Another method to linearize convective terms, applicable when using a fully coupled solution algorithm, will be presented in section 5.2.3.

4 Implicit Finite-Volume Method for Incompressible Flows – Segregated Approach

In this chapter, an algorithm, addressed to resolve the pressure-velocity coupling, is introduced. Furthermore, this chapter proposes a method of calculating mass fluxes by interpolation. This method is independent of under-relaxation. In addition, this chapter presents the detailed derivation of all coefficients that result from the discretization process. Finally, this chapter shows the boundary conditions which are relevant to the present thesis.

4.1 Discretization of the Mass Balance

Integration of equation (2.7a) over the integration domain of a single control volume P, after the application of Gauss's theorem of integration and the additivity of the Riemann integral, yields

$$\iint_{S_f} u_i n_i \mathrm{d}S = \sum_{f\in\{w,s,b,t,n,e\}} \iint_{S_f} u_i n_i \mathrm{d}S = 0.$$

In the present work, the mass balance is discretized using the midpoint integration rule for the surface integrals and linear interpolation of the velocity to the center of mass of the boundary surface S_f. These approximations lead to the following form of the mass balance

$$\sum_{f\in\{w,s,b,t,n,e\}} u_{i,f} n_{f_i} S_f = 0, \tag{4.1}$$

where no interpolation to attain the values of u_i at the face S_f has been performed yet since the straightforward linear interpolation will lead to undesired oscillations in the solution fields. Section 4.2 presents an interpolation method to circumvent this so-called *checker-boarding* effect.

4.2 A Pressure-Weighted Interpolation Method for Velocities

A cell-centered variable arrangement simplifies the treatment of non-orthogonal grids [**?**, **?**]. However, a major deficiency of cell-centered variable arrangements is that the pressure field may delink. This deficiency will then lead to unphysical oscillations in both the pressure and the velocity results. If the oscillations are severe enough, the solution algorithm might even get unstable and diverge. The described decoupling occurs, when the pressure gradient in the momentum balances, and the mass fluxes in the continuity equation are discretized using central differences.

A common practice to eliminate this behavior is the use of a momentum interpolation technique, also known as *Rhie-Chow Interpolation* [**?**]. However, the original interpolation scheme does not guarantee a unique solution, independent of the amount of under-relaxation. The performance of one of the algorithms used in the present thesis heavily relies on the under-relaxation of variables in order to accomplish stability. The present section addresses these issues by offering an interpolation method that suppresses oscillations and at the same time assures an under-relaxation independent solution: the *pressure-weighted interpolation method* [**?**, **?**, **?**, **?**, **?**].

The derivation of the pressure-weighted interpolation method builds on the discretized momentum balances at node P and an arbitrary neighboring node $Q \in NB(P)$. Section 4.5 handles the discretization used within finite-volume methods and describes the incorporation of under-relaxation factors. The semi-discrete implicit momentum balances for the velocity at node P and Q read

$$u_{i,P}^{(n)} = -\frac{\alpha_{\mathbf{u}}}{a_P^{u_i}} \left(\sum_{F\in NB(P)} a_F^{u_i} u_{i,F}^{(n)} + b_{P,u_i}^{(n-1)} - V_P \left(\frac{\partial p}{\partial x_i} \right)_P^{(n-1)} \right) + (1-\alpha_{\mathbf{u}}) u_{i,P}^{(n-1)}$$

$$\text{and} \quad u_{i,Q}^{(n)} = -\frac{\alpha_{\mathbf{u}}}{a_Q^{u_i}} \left(\sum_{F\in NB(Q)} a_F^{u_i} u_{i,F}^{(n)} + b_{Q,u_i}^{(n-1)} - V_Q \left(\frac{\partial p}{\partial x_i} \right)_Q^{(n-1)} \right) + (1-\alpha_{\mathbf{u}}) u_{i,Q}^{(n-1)} \quad .$$

Here, the superscript $(n-1)$ indicates the use of variable values of the previous outer iteration. The pressure gradient has not been discretized yet. A selective interpolation technique [**?**] can be applied. This interpolation is crucial for the prevention of the mentioned oscillations. In almost the same way, a semi-discrete implicit momentum balance can be formulated for a virtual control volume located between nodes P and Q.

$$u_{i,f}^{(n)} = -\frac{\alpha_{\mathbf{u}}}{a_f^{u_i}} \left(\sum_{F \in NB(f)} a_F^{u_i} u_{i,F}^{(n)} + b_{f,u_i}^{(n-1)} - V_f \left(\frac{\partial p}{\partial x_i} \right)_f^{(n-1)} \right) + (1-\alpha_{\mathbf{u}}) u_{i,f}^{(n-1)}. \tag{4.2}$$

Figure 4.1 gives an interpretation of the virtual control volume which shows that the motivation of this interpolation method comes form staggered variable arrangements, as figure 3.1 suggests.

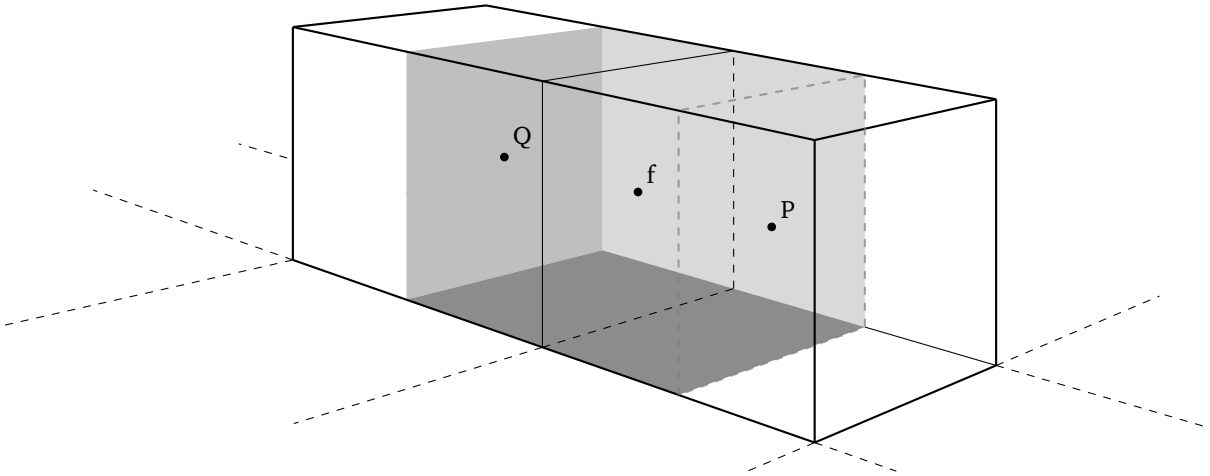


Figure 4.1: Interpretation of a virtual control volume (gray) located between nodes P and Q

To guarantee convergence of this expression for $u_{i,f}$, under-relaxation is necessary [**?**]. In order to eliminate the undefined artifacts surging from the virtualization of a control volume, the following assumptions have to be made to derive a closed expression for the velocity on the boundary face S_f:

$$\frac{\alpha_{\mathbf{u}}}{a_f^{u_i}} \left(\sum_{F \in NB(f)} a_F^{u_i} u_{i,F}^{(n)} \right) \approx \left(1-\gamma_f\right) \frac{\alpha_{\mathbf{u}}}{a_P^{u_i}} \left(\sum_{F \in NB(P)} a_F^{u_i} u_{i,F}^{(n)} \right) + \gamma_f \frac{\alpha_{\mathbf{u}}}{a_Q^{u_i}} \left(\sum_{F \in NB(Q)} a_F^{u_i} u_{i,F}^{(n)} \right), \tag{4.3a}$$

$$\frac{\alpha_{\mathbf{u}}}{a_f^{u_i}} b_{f,u_i}^{(n-1)} \approx \left(1-\gamma_f\right) \frac{\alpha_{\mathbf{u}}}{a_P^{u_i}} b_{P,u_i}^{(n-1)} + \gamma_f \frac{\alpha_{\mathbf{u}}}{a_Q^{u_i}} b_{Q,u_i}^{(n-1)} \tag{4.3b}$$

$$\text{and} \quad \frac{\alpha_{\mathbf{u}}}{a_f^{u_i}} \approx \left(1-\gamma_f\right) \frac{\alpha_{\mathbf{u}}}{a_P^{u_i}} + \gamma_f \frac{\alpha_{\mathbf{u}}}{a_Q^{u_i}}, \tag{4.3c}$$

where γ_f is a geometric interpolation factor.

By using the assumptions from equation (4.3), the expression in equation (4.2) can be closed, so that the resulting equation will only depend on the variable values in node P and Q

$$
\begin{aligned}
u_{i,f}^{(n)} \approx{}& \left(1-\gamma_f\right)\left(-\frac{\alpha_{\mathbf{u}}}{a_P^{u_i}}\sum_{F\in NB(P)} a_F^{u_i} u_{i,F}^{(n)}\right)+\gamma_f\left(-\frac{\alpha_{\mathbf{u}}}{a_Q^{u_i}}\sum_{F\in NB(Q)} a_F^{u_i} u_{i,F}^{(n)}\right)\\
&+\frac{\alpha_{\mathbf{u}}}{a_f^{u_i}} b_{f,u_i}^{(n-1)}-\frac{\alpha_{\mathbf{u}}}{a_f^{u_i}} V_f\left(\frac{\partial p}{\partial x_i}\right)_f^{(n-1)}+\left(1-\alpha_{\mathbf{u}}\right) u_{i,f}^{(n-1)}\\
={}&\left(1-\gamma_f\right) u_{i,P}^{(n)}-\left(1-\gamma_f\right)\frac{\alpha_{\mathbf{u}}}{a_P^{u_i}}\left(b_{P,u_i}^{(n-1)}-V_P\left(\frac{\partial p}{\partial x_i}\right)_P^{(n-1)}\right)+\gamma_f u_{i,Q}^{(n)}-\gamma_f\frac{\alpha_{\mathbf{u}}}{a_Q^{u_i}}\left(b_{Q,u_i}^{(n-1)}-V_Q\left(\frac{\partial p}{\partial x_i}\right)_Q^{(n-1)}\right)\\
&+\frac{\alpha_{\mathbf{u}}}{a_f^{u_i}} b_{f,u_i}^{(n-1)}-\frac{\alpha_{\mathbf{u}}}{a_f^{u_i}} V_f\left(\frac{\partial p}{\partial x_i}\right)_f^{(n-1)}+\left(1-\alpha_{\mathbf{u}}\right) u_{i,f}^{(n-1)}+\left(1-\alpha_{\mathbf{u}}\right)\left(1-\gamma_f\right) u_{i,P}+\left(1-\alpha_{\mathbf{u}}\right)\gamma_f u_{i,Q}\\
={}&\left[\left(1-\gamma_f\right) u_{i,P}^{(n)}+\gamma_f u_{i,Q}^{(n)}\right]\\
&-\left[\left(\left(1-\gamma_f\right)\frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}}+\gamma_f\frac{\alpha_{\mathbf{u}} V_Q}{a_Q^{u_i}}\right)\left(\frac{\partial p}{\partial x_i}\right)_f^{(n-1)}-\left(1-\gamma_f\right)\frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}}\left(\frac{\partial p}{\partial x_i}\right)_P^{(n-1)}-\gamma_f\frac{\alpha_{\mathbf{u}} V_Q}{a_Q^{u_i}}\left(\frac{\partial p}{\partial x_i}\right)_Q^{(n-1)}\right]\\
&+\left(1-\alpha_{\mathbf{u}}\right)\left[u_{i,f}^{(n-1)}-\left(1-\gamma_f\right) u_{i,P}^{(n-1)}-\gamma_f u_{i,Q}^{(n-1)}\right]\\
\approx{}&\left[\left(1-\gamma_f\right) u_{i,P}^{(n)}+\gamma_f u_{i,Q}^{(n)}\right]\\
&-\left(\left(1-\gamma_f\right)\frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}}+\gamma_f\frac{\alpha_{\mathbf{u}} V_Q}{a_Q^{u_i}}\right)\underline{\left[\left(\frac{\partial p}{\partial x_i}\right)_f^{(n-1)}-\left(1-\gamma_f\right)\left(\frac{\partial p}{\partial x_i}\right)_P^{(n-1)}-\gamma_f\left(\frac{\partial p}{\partial x_i}\right)_Q^{(n-1)}\right]}\\
&+\left(1-\alpha_{\mathbf{u}}\right)\left[u_{i,f}^{(n-1)}-\left(1-\gamma_f\right) u_{i,P}^{(n-1)}-\gamma_f u_{i,Q}^{(n-1)}\right].
\end{aligned}
\tag{4.4}
$$

The task of the underlined pressure gradient corrector in equation (4.4) is to suppress oscillations in the pressure field. If there are no oscillations, this part should not become active. As long as the behavior of this corrector remains consistent, i.e. that there are no oscillations in the pressure field within a converged solution, it can be multiplied by arbitrary constants [**?**]. The modification applied in the last step of the derivation of equation 4.4 makes use of this fact. The previous argumentation is only valid on equidistant grids, where $\gamma_f = 1/2$ and central differences are used to calculate the gradients. On arbitrary non-orthogonal grids, another modification has to be performed which is based on a special case of the mean value theorem of differential calculus and the following

Proposition. *Let $x_1, x_2 \in \mathbb{R}$ with $x_1 \neq x_2$ and $p(x) = a_0 + a_1 x + a_2 x^2$ a real polynomial function. Then*

$$\frac{dp}{dx}\left(\frac{x_1+x_2}{2}\right) = \frac{p(x_2)-p(x_1)}{x_2-x_1},$$

i.e. the slope of the secant equals the value of the first derivative of p exactly half the way between x_1 and x_2.

Proof. On the one hand, the evaluation of the derivative yields

$$\frac{dp}{dx}\left(\frac{x_1+x_2}{2}\right) = a_1 + 2a_2\frac{x_1+x_2}{2} = a_1 + a_2(x_1+x_2).$$

On the other hand, the slope of the secant, using the third binomial rule can be expressed as

$$\begin{aligned}\frac{p(x_2)-p(x_1)}{x_2-x_1} &= \frac{a_0+a_1x_2+a_2x_2^2-\left(a_0+a_1x_1+a_2x_1^2\right)}{x_2-x_1}\\ &= \frac{a_1(x_2-x_1)+a_2\left(x_2^2-x_1^2\right)}{x_2-x_1}\\ &= a_1+a_2(x_2+x_1).\end{aligned}$$

The comparison of both expressions completes the proof. □

It is desirable for the pressure corrector to vanish independently of the grid spacing if the profile of the pressure is quadratic and hence does not exhibit oscillations. According to the preceding proposition, this can be accomplished by modifying equation (4.4) in order to average the pressure gradients from node P and Q instead of interpolating linearly

$$\begin{aligned}u_{i,f}^{(n)} = &\left[\left(1-\gamma_f\right)u_{i,P}^{(n)}+\gamma_f u_{i,Q}^{(n)}\right]\\ &-\left(\left(1-\gamma_f\right)\frac{\alpha_{\mathbf{u}}V_P}{a_P^{u_i}}+\gamma_f\frac{\alpha_{\mathbf{u}}V_Q}{a_Q^{u_i}}\right)\left[\left(\frac{\partial p}{\partial x_i}\right)_f^{(n-1)}-\frac{1}{2}\left(\frac{\partial p}{\partial x_i}\right)_P^{(n-1)}-\frac{1}{2}\left(\frac{\partial p}{\partial x_i}\right)_Q^{(n-1)}\right]\\ &+\underline{(1-\alpha_{\mathbf{u}})\left[u_{i,f}^{(n-1)}-\left(1-\gamma_f\right)u_{i,P}^{(n-1)}-\gamma_f u_{i,Q}^{(n-1)}\right]}.\end{aligned} \tag{4.5}$$

When comparing the expression in equation (**??**) with the standard interpolation scheme, it is evident that the underlined term is not taken into consideration usually [**?**]. However, section 7.4 shows that neglecting this term indeed creates under-relaxation factor dependent results. This section concludes with a final

Proposition. *The pressure weighted momentum interpolation scheme (4.5) guarantees that the converged solution for $u_{i,f}$ is independent of the velocity under-relaxation $\alpha_{\mathbf{u}}$.*

Proof. An equivalent formulation of (4.5) is given by

$$\begin{aligned}\alpha_{\mathbf{u}}u_{i,f}^{(n-1)}+u_{i,f}^{(n-1)}-u_{i,f}^{(n)} = &\alpha_{\mathbf{u}}\left[\left(1-\gamma_f\right)u_{i,P}^{(n-1)}+\gamma_f u_{i,Q}^{(n-1)}\right]\\ &+\left[\left(1-\gamma_f\right)\left(u_{i,P}^{(n)}-u_{i,P}^{(n-1)}\right)+\gamma_f\left(u_{i,Q}^{(n)}-u_{i,Q}^{(n-1)}\right)\right]\\ &-\alpha_{\mathbf{u}}\left(\left(1-\gamma_f\right)\frac{V_P}{a_P^{u_i}}+\gamma_f\frac{V_Q}{a_Q^{u_i}}\right)\left[\left(\frac{\partial p}{\partial x_i}\right)_f^{(n-1)}-\frac{1}{2}\left(\frac{\partial p}{\partial x_i}\right)_P^{(n-1)}-\frac{1}{2}\left(\frac{\partial p}{\partial x_i}\right)_Q^{(n-1)}\right].\end{aligned}$$

Upon convergence $u_{i,P}^{(n)}\approx u_{i,P}^{(n-1)}$ and $u_{i,f}^{(n)}\approx u_{i,f}^{(n-1)}$. This leads to

$$\begin{aligned}\alpha_{\mathbf{u}}u_{i,f}^{(n-1)} = &\alpha_{\mathbf{u}}\left[\left(1-\gamma_f\right)u_{i,P}^{(n-1)}+\gamma_f u_{i,Q}^{(n-1)}\right]\\ &-\alpha_{\mathbf{u}}\left(\left(1-\gamma_f\right)\frac{V_P}{a_P^{u_i}}+\gamma_f\frac{V_Q}{a_Q^{u_i}}\right)\left[\left(\frac{\partial p}{\partial x_i}\right)_f^{(n-1)}-\frac{1}{2}\left(\frac{\partial p}{\partial x_i}\right)_P^{(n-1)}-\frac{1}{2}\left(\frac{\partial p}{\partial x_i}\right)_Q^{(n-1)}\right],\end{aligned}$$

which shows, after division by $\alpha_{\mathbf{u}}>0$, that $u_{i,f}$ is independent of the under-relaxation factor. □

4.3 Implicit Pressure-Correction and the SIMPLE-Algorithm

Finite-volume methods aim at deducing a particular system of linear algebraic equations from a partial differential equation. In the case of the momentum balances, the general structure of the resulting linear equation reads as follows

$$u_{i,P}^{(n)} = -\frac{\alpha_{\mathbf{u}}}{a_P^{u_i}}\left(\sum_{F\in NB(P)}a_F^{u_i}u_{i,F}^{(n)}+b_{P,u_i}^{(n-1)}-V_P\left(\frac{\partial p}{\partial x_i}\right)_P^{(n-1)}\right)+(1-\alpha_{\mathbf{u}})u_{i,P}^{(n-1)}.$$

Here, the pressure gradient only has been discretized symbolically and $b_{P,u_i}^{(n-1)}$ denotes the source term evaluated at the previous outer iteration.

At this stage, the equations are still coupled and nonlinear. As described in section 3.4, the Picard iteration process can be used to linearize the equations. After this, every momentum balance equation only depends on the one dominant variable u_i. Furthermore, the coupling of the momentum balances through the convective term $(u_i u_j)$ is resolved in the process of linearization. In the next step, the decoupled momentum balances can be solved sequentially for the dominant variable u_i. All coefficients $a_{\{P,F\}}^{u_i}$, the source term, and the pressure gradient will be evaluated explicitly by using results of the preceding outer iteration $(n-1)$. For the pressure gradient, this means to take the pressure of the antecedent outer iteration as an approximate for the following iteration. This approximate has to be corrected in an iterative way until all the nonlinear equations are fulfilled up to a specified tolerance. Section (6.3.2) presents a suitable convergence criterion and its implementation. This linearization process in conjunction with the pressure approximate leads to the linear equation

$$u_{i,P}^{(n*)} = -\frac{\alpha_{\mathbf{u}}}{a_P^{u_i}} \left(\sum_{F \in NB(P)} a_F^{u_i} u_{i,F}^{(n*)} + b_{P,u_i}^{(n-1)} - V_P \left(\frac{\partial p}{\partial x_i} \right)_P^{(n-1)} \right) + (1-\alpha_{\mathbf{u}}) u_{i,P}^{(n-1)}. \tag{4.6}$$

Here, $(*)$ indicates that in order to fulfill the discretized mass balance

$$\sum_{F \in NB(P)} u_{i_f}^{(n)} n_i S_f = 0. \tag{4.7}$$

the solution of this equation still needs to be corrected. Applying the procedure from section 4.2 to equation (4.6) results in the following expression for the face velocities after solving the discretized momentum balances

$$\begin{aligned} u_{i,f}^{(n*)} = & \left[(1-\gamma_f) u_{i,P}^{(n*)} + \gamma_f u_{i,Q}^{(n*)} \right] \\ & - \left((1-\gamma_f) \frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}} + \gamma_f \frac{\alpha_{\mathbf{u}} V_Q}{a_Q^{u_i}} \right) \left[\left(\frac{\partial p}{\partial x_i} \right)_f^{(n-1)} - (1-\gamma_f) \left(\frac{\partial p}{\partial x_i} \right)_P^{(n-1)} - \gamma_f \left(\frac{\partial p}{\partial x_i} \right)_Q^{(n-1)} \right] \\ & + (1-\alpha_{\mathbf{u}}) \left[u_{i,f}^{(n-1)} - (1-\gamma_f) u_{i,P}^{(n-1)} - \gamma_f u_{i,Q}^{(n-1)} \right]. \end{aligned} \tag{4.8}$$

Here, the pressure from the previous outer iteration has been used as a pressure approximate for the pressure gradients.

The lack of an equation having the pressure as dominant variable leads to the necessity of altering the mass balance as the only equation left from the set of partial differential equations (2.7). Methods of this type are called projection methods [**?**]. A common class of algorithms of this family of methods uses an equation for the additive pressure-correction p' instead of the pressure itself. These algorithms enforce continuity by additively correcting the velocities and the pressure with the correctors u_i' and p' to

$$u_{i,P}^{(n)} = u_{i,P}^{(n*)} + u_{i,P}', \quad u_{i,f}^{(n)} = u_{i,f}^{(n*)} + u_{i,f}' \quad \text{and} \quad p_P^{(n)} = p_P^{(n-1)} + p_P'.$$

Similar expressions exist for the variable values located at node Q. Now it is possible to formulate the discretized momentum balance for the corrected velocities and the corrected pressure as

$$u_{i,P}^{(n)} = -\frac{\alpha_{\mathbf{u}}}{a_P^{u_i}} \left(\sum_{F \in NB(P)} a_F^{u_i} u_{i,F}^{(n)} + b_{P,u_i}^{(n-1)} - V_P \left(\frac{\partial p}{\partial x_i} \right)_P^{(n)} \right) + (1-\alpha_{\mathbf{u}}) u_{i,P}^{(n-1)}. \tag{4.9}$$

The only difference to the equation which will be solved in the next outer iteration is that the source term b_{P,u_i} has not been updated yet. In the same way, the discretized momentum balance for the face velocity $u_{i,f}$ can be formulated as

$$\begin{aligned} u_{i,f}^{(n)} = & \left[(1-\gamma_f) u_{i,P}^{(n)} + \gamma_f u_{i,Q}^{(n)} \right] \\ & - \left((1-\gamma_f) \frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}} + \gamma_f \frac{\alpha_{\mathbf{u}} V_Q}{a_Q^{u_i}} \right) \left[\left(\frac{\partial p}{\partial x_i} \right)_f^{(n)} - (1-\gamma_f) \left(\frac{\partial p}{\partial x_i} \right)_P^{(n)} - \gamma_f \left(\frac{\partial p}{\partial x_i} \right)_Q^{(n)} \right] \\ & + (1-\alpha_{\mathbf{u}}) \left[u_{i,f}^{(n-1)} - (1-\gamma_f) u_{i,P}^{(n-1)} - \gamma_f u_{i,Q}^{(n-1)} \right]. \end{aligned} \tag{4.10}$$

To couple velocity and pressure correctors one can subtract equation (4.6) from (4.9) and equation (4.8) from (4.10) to get

$$u'_{i,P} = -\frac{\alpha_{\mathbf{u}}}{a_P^{u_i}} \left(\underline{\sum_{F \in NB(P)} a_F^{u_i} u'_{i,F}} - V_P \left(\frac{\partial p'}{\partial x_i} \right)_P^{(n)} \right) \quad \text{and} \tag{4.11}$$

$$u'_{i,f} = \left[(1-\gamma_f) u'_{i,P} + \gamma_f u'_{i,Q} \right] - \left((1-\gamma_f) \frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}} + \gamma_f \frac{\alpha_{\mathbf{u}} V_Q}{a_Q^{u_i}} \right) \left[\left(\frac{\partial p}{\partial x_i} \right)'_f - (1-\gamma_f) \left(\frac{\partial p}{\partial x_i} \right)'_P - \gamma_f \left(\frac{\partial p}{\partial x_i} \right)'_Q \right]. \tag{4.12}$$

The majority of the class of pressure-correction algorithms has these equations as a common basis. Each algorithm then introduces particular, distinguishable approximations of the velocity corrections that are, at the moment of solving for the pressure-correction, still unknown. The method used in the present work is the *SIMPLE*-algorithm (*Semi-Implicit Method for Pressure-Linked Equations*) [**?**]. The approximation which this algorithm performs is severe since the term containing the unknown velocity corrections is dropped entirely. The respective term has been underlined in equation (4.11). Since the global purpose of the presented method is to enforce continuity by implicitly calculating a pressure-correction, the velocity correction has to be expressed explicitly in terms of the pressure-correction. This can be accomplished by inserting equation (4.11) into equation (4.12). This approach gives an update formula

$$u'_{i,f} \approx -\left((1-\gamma_f) \frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}} + \gamma_f \frac{\alpha_{\mathbf{u}} V_Q}{a_Q^{u_i}} \right) \left(\frac{\partial p}{\partial x_i} \right)'_f, \tag{4.13}$$

which is then, together with (4.8), inserted into the discretized continuity equation (4.7) to obtain

$$\sum_{F \in NB(P)} \left((1-\gamma_f) \frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}} + \gamma_f \frac{\alpha_{\mathbf{u}} V_F}{a_F^{u_i}} \right) \left(\frac{\partial p}{\partial x_i} \right)'_f n_i S_f = b_{P,p} \quad . \tag{4.14}$$

Here, the right-hand side $b_{P,p}$ is defined as

$$b_{P,p} := \sum_{F \in NB(P)} u_{i,f}^{(n*)} n_i S_f. \tag{4.15}$$

The complete discretization of this equation, using central differences as approximation for the gradient of the pressure-correction, is straightforward and will be presented in section 4.4.

The approximation performed in the SIMPLE-algorithm affects convergence in a way that the pressure-correction has to be under-relaxed with a parameter $\alpha_p \in (0,1]$

$$p_P^{(n)} = p_P^{(n-1)} + \alpha_{\mathbf{p}} p'_P. \tag{4.16}$$

There exist better approximations for the pressure-correction which do not rely on pressure under-relaxation for convergence. On example that uses a more consistent approximation is the *SIMPLEC*-algorithm (*SIMPLE Consistent*) [**?**]. [**?**] gives a similar derivation of the pressure weighted interpolation method and its implementation in the SIMPLEC-algorithm.

As shown in section 4.2, the behavior of the pressure weighted interpolation method on non-equidistant grids can be improved by replacing the linear interpolation of pressure gradients in equation (4.8) by unweighted arithmetic averaging. This replacement leads to the following equation for calculating mass fluxes from face velocities

$$\begin{aligned} u_{i,f}^{(n*)} = & \left[(1-\gamma_f) u_{i,P}^{(n*)} + \gamma_f u_{i,Q}^{(n*)} \right] \\ & - \left((1-\gamma_f) \frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}} + \gamma_f \frac{\alpha_{\mathbf{u}} V_Q}{a_Q^{u_i}} \right) \left[\left(\frac{\partial p}{\partial x_i} \right)_f^{(n-1)} - \frac{1}{2} \left(\frac{\partial p}{\partial x_i} \right)_P^{(n-1)} - \frac{1}{2} \left(\frac{\partial p}{\partial x_i} \right)_Q^{(n-1)} \right] \\ & + (1-\alpha_{\mathbf{u}}) \left[u_{i,f}^{(n-1)} - (1-\gamma_f) u_{i,P}^{(n-1)} - \gamma_f u_{i,Q}^{(n-1)} \right]. \end{aligned} \tag{4.17}$$

The SIMPLE-algorithm can be represented by an iterative procedure as shown in Algorithm 1. If the solution algorithm includes a decoupled variable, i.e. a passively convected quantity it is preferable to solve for this variable after the

application of the entire iterative solution algorithm for the velocity and pressure field. This approach reduces the number of linear solves for the scalar equation.

Algorithm 1 SIMPLE-Algorithm

INITIALIZE variables
while (convergence criterion not accomplished) **do**
  SOLVE linearized momentum balances **(4.6)**
  CALCULATE mass fluxes using **(4.10)** or **(4.17)**
  SOLVE pressure-correction equation **(4.14)**to assure continuity
  UPDATE pressure using **(4.16)**
  UPDATE velocities and mass fluxes using **(4.11)**
  if (coupled scalar equation) **then**
    SOLVE scalar equation as described in section 4.6
  end if
end while
if (decoupled scalar equation) **then**
  SOLVE scalar equation as described in section 4.6
end if

4.4 Discretization of the Mass Fluxes and the Pressure-Correction Equation

Sections 4.2 and 4.3 introduced the concept of pressure weighted interpolation to avoid oscillating results and an algorithm to calculate a velocity field that obeys continuity. The derived equations have not been discretized completely. Furthermore, the approach has not been generalized to non-orthogonal grids.

The discretized mass balance (4.1) only depends on the normal velocities $u_{i,f}\,n_{i,f}$. By analogy with equation (4.17), an interpolated normal face velocity, and thus the mass flux, can be calculated as follows

$$\begin{aligned} u_{i,f}^{(n*)} n_{i,f} = &\Big[\big(1-\gamma_f\big)u_{i,P}^{(n*)} + \gamma_f u_{i,Q}^{(n*)}\Big] n_{i,f} \\ &-\left(\big(1-\gamma_f\big)\frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}} + \gamma_f \frac{\alpha_{\mathbf{u}} V_Q}{a_Q^{u_i}}\right)\left[\left(\frac{\partial p}{\partial n}\right)_f^{(n-1)} - \frac{1}{2}\left(\frac{\partial p}{\partial n}\right)_P^{(n-1)} - \frac{1}{2}\left(\frac{\partial p}{\partial n}\right)_Q^{(n-1)}\right] \\ &+ (1-\alpha_{\mathbf{u}})\Big[u_{i,f}^{(n-1)} - \big(1-\gamma_f\big)u_{i,P}^{(n-1)} - \gamma_f\, u_{i,Q}^{(n-1)}\Big] n_{i,f}. \end{aligned} \tag{4.18}$$

Here, the scalar product of the pressure gradient and the normal vector has been replaced by a directional derivative in the direction of the face unit normal vector. In the present work, the pressure gradients in (4.18) and the pressure-correction gradients in equation (4.14) will be discretized by central differences

$$\left(\frac{\partial p}{\partial n}\right)_f \approx \frac{p_P - p_Q}{\big(\mathbf{x}_P - \mathbf{x}_Q\big)\cdot \mathbf{n}_f} \quad \text{and} \quad \left(\frac{\partial p'}{\partial n}\right)_f \approx \frac{p'_P - p'_Q}{\big(\mathbf{x}_P - \mathbf{x}_Q\big)\cdot \mathbf{n}_f}.$$

In the next step, this discretization can be inserted into the semi-discretized pressure-correction equation (4.14)

$$\sum_{F\in NB(P)} \left(\big(1-\gamma_f\big)\frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}} + \gamma_f \frac{\alpha_{\mathbf{u}} V_F}{a_F^{u_i}}\right)\frac{p'_P - p'_F}{(\mathbf{x}_P - \mathbf{x}_F)\cdot \mathbf{n}_f} S_f = b_{P,p}.$$

The resulting coefficients for the pressure-correction equation,

$$a_P^{p'} p'_P + \sum_{F\in NB(P)} a_F^{p'} p'_F = b_{P,p'},$$

can be calculated as follows

$$a_F^{p'} = -\left(\big(1-\gamma_f\big)\frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}} + \gamma_f \frac{\alpha_{\mathbf{u}} V_F}{a_F^{u_i}}\right)\frac{S_f}{(\mathbf{x}_P - \mathbf{x}_F)\cdot \mathbf{n}_f} \quad \text{and} \quad a_P^{p'} = -\sum_{F\in NB(P)} a_F^{p'}. \tag{4.19}$$

The right-hand side can be calculated as in equation (4.15) if the presented discretization is applied.

4.5 Discretization of the Momentum Balance

If the stationary momentum balance is integrated over a single control volume P, the result reads

$$\underbrace{\iint_S (\rho u_i u_j) n_j \mathrm{d}S}_{\text{convective term}} - \underbrace{\iint_S \left(\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right) n_j \mathrm{d}S}_{\text{diffusive term}} = \underbrace{-\iiint_V \frac{\partial p}{\partial x_i}\mathrm{d}V}_{\text{pressure source term}} \underbrace{-\iiint_V \rho\beta\,(T - T_0)\, g_i \mathrm{d}V}_{\text{temperature source term}}. \tag{4.20}$$

Here, the different terms are indicated which will be addressed individually in the following subsections. Due to the stationarity of the flow problems to be solved the terms of equations (2.7) involving time derivatives have been ignored. The form of equation (4.20) has been derived by using Gauss's theorem of integration. The terms residing on the left will be dealt with in an implicit and, due to deferred corrections, also in an explicit way, whereas the terms on the right will be treated exclusively in an explicit way.

4.5.1 Linearization and Discretization of the Convective Term

The convective term $\left(\rho u_i u_j\right)$ of the Navier-Stokes equations is the reason for the nonlinearity of the equations. In order to deduce a set of linear algebraic equations from the Navier-Stokes equations, this term has to be linearized. As introduced in section (3.4), the Picard iteration process will address the nonlinearity. Therefore, the part dependent on the non-dominant dependent variable, will be approximated with its value from the previous iteration as $\rho u_i^{(n)} u_j^{(n)} \approx \rho u_i^{(n)} u_j^{(n-1)}$. However, this linearization will not be directly visible, because it will be covered by the mass flux $\dot{m}_f = \iint_{S_f} \rho u_j^{(n-1)} n_j \mathrm{d}S$.

Using the additivity of the Riemann integral, the first step is to decompose the surface integral into individual contributions from each boundary face of the control volume P

$$\iint_S \left(\rho u_i u_j\right) n_j \mathrm{d}S = \sum_{f\in\{w,s,b,t,n,e\}} \iint_{S_f} \left(\rho u_i u_j\right) n_j \mathrm{d}S = \sum_{f\in\{w,s,b,t,n,e\}} F^c_{i,f},$$

where $F^c_{i,f} := \iint_{S_f} \left(\rho u_i^{(n)} u_j^{(n-1)}\right) n_j \mathrm{d}S$ is the convective flux of the velocity u_i through the boundary face S_f.

To improve diagonal dominance of the resulting linear system, while maintaining the smaller discretization error of a higher order discretization, a blended discretization scheme is combined with a deferred correction. Since, due to the nonlinearity of the equations to be solved, an iterative solution process is needed by all means, the overall convergence does not degrade noticeably by the application of a deferred correction [**?**]. Blending and deferred correction result in a decomposition of the convective flux into a lower order approximation, which is treated implicitly, and an explicit difference between the higher and lower order approximation for the same convective flux. The use of higher order approximations on coarse grids may lead to oscillations in the solution. This fact in turn may deteriorate or even impede convergence. In order to prevent this degradation of convergence, a control factor $\eta \in [0,1]$ can blend implicit and explicit schemes. Furthermore, the use of low-order approximations increases the diagonal dominance of the matrix, which in turn can improve the convergence of certain iterative solvers for the linear systems [**?**].

To show the generality of this approach, all further derivations are presented for the generic boundary face S_f which separates control volume P from its neighbor $F \in NB(P)$. The fluxes can be decomposed as follows

$$F^c_{i,f} \approx \underbrace{F^{c,l}_{i,f}}_{\text{implicit}} + \eta \underbrace{\left[F^{c,h}_{i,f} - F^{c,l}_{i,f}\right]^{(n-1)}}_{\text{explicit}}. \tag{4.21}$$

The convective fluxes carrying an l for *lower* or an h for *higher* as exponent, already have been linearized and discretized. The discretization applied to the convective flux in the present work is using the midpoint integration rule and blends

the upwind interpolation scheme with a linear interpolation scheme. Using the decomposition from equation (4.21), one can derive the following approximations

$$F_{i,f}^{c,l} = u_{i,F}\min(\dot{m}_f, 0) + u_{i,P}\max(0, \dot{m}_f)$$
$$F_{i,f}^{c,h} = \dot{m}_f\,\gamma_f\,u_{i,F} + \dot{m}_f\,(1-\gamma_f)\,u_{i,P},$$

where, the variable values may have to be taken from the previous iteration step $(n-1)$ as described in equation (4.21) and the mass flux $\dot{m}_f$ has been calculated as result of the linearization process. Now, the results are summarized by presenting the convective contribution to the matrix coefficients $a_F^{u_i}$ and $a_P^{u_i}$ and the right-hand side b_{P,u_i} as

$$a_F^{u_i,c} = \min(\dot{m}_f, 0), \qquad a_P^{u_i,c} = \sum_{F\in NB(P)} \max(0, \dot{m}_f)$$

$$\text{and} \quad b_{P,u_i}^{c} = \sum_{F\in NB(P)} \eta\left(u_{i,F}^{(n-1)}\left(\min(\dot{m}_f, 0) - \dot{m}_f\gamma_f\right)\right) + \eta\left(u_{i,P}^{(n-1)}\left(\max(0, \dot{m}_f) - \dot{m}_f\left(1-\gamma_f\right)\right)\right).$$

4.5.2 Discretization of the Diffusive Term

The diffusive term contains the first partial derivatives of the velocity as a result of the material constitutive equation that characterizes the behavior of Newtonian fluids. As section 3.3 points out, directional derivatives can be discretized using central differences on orthogonal grids. In the universal case of non-orthogonal grids, the discretization can be carried out using implicit central differences and an explicit deferred correction which addresses the non-orthogonality of the grid. As seen in equation (2.6), the diffusive term of the Navier-Stokes equations can be simplified using the mass balance for the case of an incompressible flow with constant viscosity μ. In order to preserve the general validity of the presented approach, this simplification will be omitted.

As before, by using the additivity and furthermore linearity of the Riemann integral, the integration of the diffusive term will be divided into integrations over the individual boundary faces S_f

$$\iint_S \left(\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right) n_j \mathrm{d}S = \sum_{f\in\{w,s,b,t,n,e\}} \left[\iint_{S_f} \mu\underline{\frac{\partial u_i}{\partial x_j} n_j \mathrm{d}S} + \iint_{S_f} \mu\frac{\partial u_j}{\partial x_i} n_j \mathrm{d}S\right] = \sum_{f\in\{w,s,b,t,n,e\}} F_{i,f}^{d}.$$

Here, $F_{i,f}^{d}$ denotes the diffusive flux through an individual boundary face. Section 3.3 only covered the non-orthogonal corrector for directional derivatives. Since the velocity is a vector field and not a scalar field, the results of section 3.3 may only be applied to the underlined term. The other term will be treated explicitly since its sum over all boundary faces is considerably smaller than the underlined term. Furthermore, the explicit term does not cause oscillations and thus will not deteriorate convergence [**?**]. All present integrals will be approximated using the midpoint integration rule. The diffusive flux $F_{i,f}^{d}$ for the generic face S_f, located between the control volumes P and F, reads as follows

$$F_{i,f}^{d} \approx \underline{\mu\left(\frac{\partial u_i}{\partial x_j}\right)_f n_j S_f} + \mu\left(\frac{\partial u_j}{\partial x_i}\right)_f n_j S_f.$$

By using central differences for the implicit discretization of the directional derivative and by using the *orthogonal correction* approach from section 3.3.2, the approximation can be derived as follows

$$F_{i,f}^{d} \approx \mu\left(\underline{||\boldsymbol{\Delta}_f||_2 \frac{u_{i,P} - u_{i,F}}{||\mathbf{x}_P - \mathbf{x}_F||_2} - (\nabla u_i)_f^{(n-1)} \cdot \left(\boldsymbol{\Delta}_f - \mathbf{S}_f\right)}\right) + \mu\left(\frac{\partial u_j}{\partial x_i}\right)_f^{(n-1)} n_{f_i}$$

$$= \mu\left(\underline{S_f \frac{u_{i,P} - u_{i,F}}{||\mathbf{x}_P - \mathbf{x}_F||_2} - \left(\frac{\partial u_i}{\partial x_j}\right)_f^{(n-1)} \left(\xi_{f_i} - n_{f_i}\right) S_f}\right) + \mu\left(\frac{\partial u_j}{\partial x_i}\right)_f^{(n-1)} n_{f_i}.$$

Here, the unit vector pointing $\boldsymbol{xi}_f = \left(\xi_{f\,i}\right)_{i=1,\ldots,3}$ in the direction of the straight line, which connects control volume P and control volume F is denoted as follows

$$\xi_f = \frac{\mathbf{x}_P - \mathbf{x}_F}{||\mathbf{x}_p - \mathbf{x}_F||_2}.$$

The interpolation of the cell center gradients to the boundary faces is performed as in equation (3.2). Now, the contribution of the diffusive part to the matrix coefficients and the right-hand side can be calculated as

$$a_F^{u_i,d} = -\frac{\mu S_f}{||\mathbf{x}_P - \mathbf{x}_F||_2}, \qquad a_P^{u_i,d} = \sum_{F \in NB(P)} \frac{\mu S_f}{||\mathbf{x}_P - \mathbf{x}_F||}$$

$$\begin{aligned} b_{F,u_i}^d &= -\sum_{F \in NB(P)} \left(\frac{\partial u_i}{\partial x_j}\right)_f^{(n-1)} \left(\xi_{f_i} + n_{f_i}\right) S_f - \mu \left(\frac{\partial u_j}{\partial x_i}\right)_f^{(n-1)} n_{f_i} S_f \\ &= -\sum_{F \in NB(P)} \left(\frac{\partial u_i}{\partial x_j}\right)_f^{(n-1)} \xi_{f_i} S_f + \mu \left(\left(\frac{\partial u_i}{\partial x_j}\right)_f^{(n-1)} + \left(\frac{\partial u_j}{\partial x_i}\right)_f^{(n-1)}\right) n_{f_i} S_f . \end{aligned}$$

4.5.3 Discretization of the Source Terms

Since in the segregated solution approach in every equation all other variables but the dominant one are treated as constants, and the source terms in equation (4.20) do not depend on the dominant variable, the discretization is straightforward. The source terms of the momentum balance are discretized using the midpoint integration rule for volume integrals. This discretization leads to the source term

$$-\iiint_V \frac{\partial p}{\partial x_i} \mathrm{d}V - \iiint_V \rho \beta \left(T - T_0\right) g_i \mathrm{d}V \approx -\left(\frac{\partial p}{\partial x_i}\right)_P^{(n-1)} V_P - \rho \beta \left(T_P^{(n-1)} - T_0\right) g_i V_P = b_{P,u_i}^{sc}.$$

4.6 Discretization of the Temperature Equation

The discretization of the stationary temperature equation is performed by the same means as for the momentum balance. The only difference is a simpler diffusion term. Having applied Gauss's theorem of integration, the integral form of the temperature equation reads as follows

$$\underbrace{\iint_S \rho u_j T n_j \mathrm{d}S}_{\text{advective term}} - \underbrace{\iint_S \kappa \frac{\partial T}{\partial x_j} n_j \mathrm{d}V}_{\text{diffusive term}} = \underbrace{\iiint_V q_T \mathrm{d}V}_{\text{source term}}.$$

Now, proceeding as in the preceding sections, one can discretize the advective term, the diffusive term, and the source term. Since this process does not provide further insight, only the final results will be presented. The discretization yields the matrix coefficients as

$$a_F^T = \min(\dot{m}_f, 0) + \frac{\kappa S_f}{||\mathbf{x}_P - \mathbf{x}_F||_2}$$

$$a_P^T = \sum_{F \in NB(P)} \max(0, \dot{m}_f) - \frac{\kappa S_f}{||\mathbf{x}_P - \mathbf{x}_F||_2}$$

$$\begin{aligned} b_{P,T} = & \sum_{F \in NB(P)} \eta \left(T_F^{(n-1)} \left(\min(\dot{m}_f, 0) - \dot{m}_f \gamma_f \right) \right) + \eta \left(T_P^{(n-1)} \left(\max(0, \dot{m}_f) - \dot{m}_f \left(1 - \gamma_f \right) \right) \right) \\ & + \sum_{F \in NB(P)} \kappa \left(\frac{\partial T}{\partial x_j} \right)_f^{(n-1)} \left(\xi_{f_j} - n_{f_j} \right) S_f \\ & + q_{T_P} V_P. \end{aligned}$$

Again it is possible though not always as necessary as in the case of the velocities, to under-relax the solution of the resulting linear system with an under-relaxation factor α_T. Section 4.9 shows how to accomplish implicit under-relaxation of variables.

4.7 Boundary Conditions

As the antecedent sections showed, it is possible to deduce a linear algebraic equation for each control volume by the finite-volume method. However, the approach presented in the preceding sections did not cover the treatment of control volumes at domain boundaries. Hence, this section introduces the boundary conditions which are relevant for the present work and deals with transitional conditions at block boundaries.

4.7.1 Dirichlet Boundary Conditions

The first boundary condition is the Dirichlet boundary condition. This type of boundary condition, when used for the velocity or transported scalars, is used to model inlet conditions for flow problems. For the temperature equation, it may also be used at walls, as will be shown in subsection 4.7.2. The Dirichlet boundary condition is characterized by explicitly specifying the value of the variable for which the equation is solved. As a result, boundary fluxes can be calculated directly. In the case of Dirichlet boundary conditions for the velocities, especially the mass flux $\dot{m}_f$ is known and hence does not have to be calculated using the pressure weighted interpolation method. This fact furthermore implies that the boundary velocities do not have to be corrected at Dirichlet velocity boundaries, i.e. equation (4.13) yields

$$0 = u'_{i,f} = -\left(\left(1 - \gamma_f \right) \frac{\alpha_{\mathbf{u}} V_P}{a_P^{u_i}} + \gamma_f \frac{\alpha_{\mathbf{u}} V_Q}{a_Q^{u_i}} \right) \left(\frac{\partial p}{\partial x_i} \right)'_f$$

which implies

$$\left(\frac{\partial p}{\partial x_i} \right)'_f = 0. \tag{4.22}$$

Equation (4.22) shows that a Neumann zero gradient boundary condition applies to the pressure-correction at Dirichlet velocity boundaries.

In the case of Dirichlet boundary conditions for the velocities or the temperature, the coefficient corresponding to an imaginary neighboring control volume laying past the boundary is considered as an explicit contribution to the right-hand side of the linear system. Since no modifications to the discretized equations have to be considered, the implementation approach for inlet boundary conditions will be presented for the temperature equation only. At Dirichlet boundaries the Furthermore, there is no boundary condition that fixes the gradient at Dirichlet boundaries. In this case, it is common to assume that the partial derivatives of the respective variable are constant and can be extrapolated as follows

$$\left(\frac{\partial T}{\partial x_j} \right)_f \approx \left(\frac{\partial T}{\partial x_j} \right)_P.$$

The modification of the central coefficient of the linear equation can be formulated recursively as

$$a_P^T = a_P^T + \left(\max(0, \dot{m}_f) - \frac{\kappa S_f}{||\mathbf{x}_P - \mathbf{x}_f||_2} \right), \tag{4.23}$$

whereas the contribution to the right-hand side reads as follows

$$b_{P,T} = b_{P,T} - \left(\min(\dot{m}_f, 0) - \frac{\kappa S_f}{||\mathbf{x}_P - \mathbf{x}_f||_2} \right) T_f + \left(\frac{\partial T}{\partial x_j} \right)_P^{(n-1)} \left(\xi_{f_j} - n_{f_j} \right) S_f . \tag{4.24}$$

Equations (4.23) and (4.24) show, that a blending factor $\eta = 0$ is used. Even though the gradient discretization at domain boundaries is realized by a one-sided forward differencing scheme instead of a central differencing scheme, this does not drastically affect accuracy, because the distance used in the differential quotient is about half the distance used within a central difference inside the domain [**?**].

The case of Dirichlet boundary conditions for the pressure needs special consideration since no equation for the pressure is solved but instead an equation for the pressure-correction. This type of boundary condition can be applied to inlet boundaries or pressure boundaries and is recommendable if, on the one hand, the velocity distribution is unknown, but, on the other hand, the pressure difference between inlet and outlet, or the reference pressure at the outlet is a known quantity. The pressure-correction at Dirichlet pressure boundaries is assumed to $p' = 0$, and the modification of the coefficients of the algebraic equation for the corresponding control volumes can be carried out as in equations (4.23) and (4.24). The velocity correction at the boundaries is nonzero and will have to be extrapolated using the formula for the pressure weighted interpolation applied to boundary values (4.12). Even though Dirichlet boundary conditions for the pressure would correspond to Neumann boundary conditions for the velocities, a Dirichlet boundary condition for the velocities is used as well, using an extrapolated velocity value as fixed value for the calculation of boundary fluxes in the momentum balances.

4.7.2 Treatment of Wall Boundaries

Solid, impermeable walls represent a common boundary for the solution domain. This boundary condition first of all has a kinematic character since the concept of impermeable walls dictates a zero normal velocity at the wall. In viscous flows, wall boundaries can be interpreted furthermore as a no-slip condition, i.e. a Dirichlet boundary condition for the tangential velocities. Convective fluxes through solid walls are thus zero by definition. However, the diffusive fluxes of the velocities and the temperature require special consideration. The correct approximation of fluid behavior on a wall boundary requires modifying the diffusive terms of the momentum balance. Furthermore, diffusive fluxes for the temperature can be given by Neumann or Dirichlet boundary conditions.

The derivation of the discretized diffusive flux through wall boundaries starts from the integral momentum balance (2.2) for the vector $\mathbf{u}$. Here, only the term for surface forces is needed

$$\mathbf{F}_w = \iint_{S_w} \mathbf{t} \mathrm{d}S = \iint_{S_w} \mathbf{T}(\mathbf{n}_w) \mathrm{d}S.$$

For the purpose of treating wall boundary conditions, it is appropriate to use a local coordinate system n, t, s, where n denotes the wall-normal coordinate, t denotes the coordinate tangential to the wall shear force and, s is the binormal coordinate. With respect to this coordinate system, the wall-normal vector is represented by $\mathbf{n}_w = (1,0,0)^T$ and the image of the wall-normal vector $\mathbf{T}(\mathbf{n_w})$ is represented by

$$\mathbf{T}(\mathbf{n}_w) = \begin{bmatrix} \tau_{nn} & \tau_{nt} & \tau_{ns} \\ \tau_{nt} & \tau_{tt} & \tau_{ts} \\ \tau_{ns} & \tau_{ts} & \tau_{ss} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \tau_{nn} \\ \tau_{nt} \\ \tau_{ns} \end{bmatrix}.$$

The velocity of the wall is assumed to be constant [**?**], what implies that the directional derivative of the tangential velocity vanishes on the wall

$$\tau_{tt} = \frac{\partial u_t}{\partial x_t} = 0,$$

which, together with the continuity equation in differential form (2.1), leads to

$$\frac{\partial u_n}{\partial x_n} + \frac{\partial u_t}{\partial x_t} = \frac{\partial u_n}{\partial x_n} = 0, \tag{4.25}$$

what is equivalent to $\tau_{nn} = 0$ at the wall. The velocity is assumed to have no component in the binormal direction. A physical interpretation of (4.25) would be that the transfer of momentum at the wall occurs by shear forces exclusively. Furthermore, the coordinate direction t is chosen to be parallel to the shear force. This choice does not represent a restriction, because of the possibility to rotate the coordinate system within the plane. This choice of t leads to $\tau_{ns} = 0$. Hence, the absolute value of the surface force only depends on the normal derivative of the velocity tangential to the wall. After transforming the coordinates back to the system (x_1, x_2, x_3) the surface force can be calculated, and the integral can be discretized using the midpoint integration rule

$$\mathbf{F}_w = \iint_{S_w} \mathbf{t_w}\, \tau_{nt} \mathrm{d}S = \iint_{S_w} \mathbf{t_w}\, \mu \frac{\partial u_t}{\partial n} \mathrm{d}S = \mathbf{t_w}\, \mu \left(\frac{\partial u_t}{\partial n}\right)_w S_w, \tag{4.26}$$

where $\mathbf{t_w}$ denotes the transformed tangential vector $(0,1,0)^T$ with respect to the coordinate system (x_1, x_2, x_3). In the discretization process, this tangential vector will be calculated from the velocity vector as

$$\mathbf{t_w} = \frac{\mathbf{u}_t}{||\mathbf{u}_t||_2} \quad \text{, where} \quad \mathbf{u}_t = \mathbf{u} - (\mathbf{u} \cdot \mathbf{n}_w)\, \mathbf{n}_w.$$

According to [**?**] this force should not be handled explicitly for convergence reasons. If surface forces are expressed in terms of the velocities u_i, the discretization process leads to different central matrix coefficients, which in turn deteriorates memory efficiency.

The discretization approach used in the present thesis uses a simpler implicit discretization coupled with a deferred correction which combines the explicit discretization of the surface force (4.26) with a central difference. At first, the contained directional derivative is discretized implicitly by

$$\left(\frac{\partial u_t}{\partial n}\right) t_{wi} \approx \left(\frac{\partial u_i}{\partial \xi}\right) \approx \frac{u_{i,P} - u_{i,w}}{||\mathbf{x}_P - \mathbf{x}_w||_2}$$

and explicitly by

$$\left(\frac{\partial u_t}{\partial n}\right) t_{wi} \approx \frac{\left(u_{i,P} - u_{i,w}\right) - \left(u_{j,P} - u_{j,w}\right) n_j n_i}{(\mathbf{x}_P - \mathbf{x}_w) \cdot \mathbf{n}_w}.$$

Therefore, the contributions to the central coefficient and the right-hand side of the linear equation that results from the presented discretization process read

$$a_P^{u_i} = a_P^{u_i} + \mu \frac{S_f}{||\mathbf{x}_P - \mathbf{x}_w||_2} \quad \text{and}$$

$$b_{P,u_i} = b_P^{u_i} + \mu \frac{S_f}{||\mathbf{x}_P - \mathbf{x}_w||_2} u_{i,P}^{(n-1)} + \mu \frac{\left(u_{i,P}^{(n-1)} - u_{i,w}\right) - \left(u_{j,P}^{(n-1)} - u_{j,w}\right) n_j n_i}{(\mathbf{x}_P - \mathbf{x}_w) \cdot \mathbf{n}_w}.$$

Since the deferred correction uses a Dirichlet boundary condition, no correction of the value of the wall velocity $u_{i,w}$ has to be accounted for.

If the solution of the flow field is coupled to the solution of a temperature equation, different options for the boundary condition at walls may be chosen. If, on the one hand, the wall temperature is known, a Dirichlet boundary condition for the temperature is the choice. A wall of this type is called to be *isothermal*. If, on the other hand, only the heat flux is known, a Neumann boundary condition is used. In the special case of zero heat flux, the wall is called to be *adiabatic*. Besides isothermal walls, the present thesis uses adiabatic walls. The implementation of isothermal walls is straightforward since no new coefficients for the temperature equation have to be calculated.

4.7.3 Treatment of Block Boundaries

If structured grids are used to decompose the problem domain, the characterizing property of structured grids, which is the constant amount of grid cells in each direction, gives each inner cell exactly six neighbors. If block-structured grids

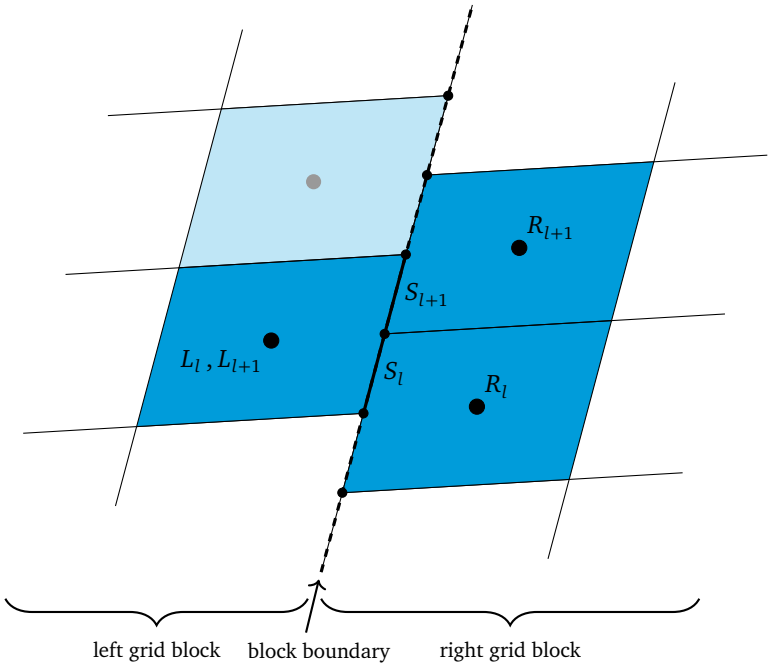


Figure 4.2: Non-matching grid cells with hanging nodes at a two-dimensional block boundary. Indexing is based on the face segments S_l

are used, this property may be violated if the number of grid cells of each block is chosen to be arbitrary. The arbitrariness of the grid resolution is a main benefactor for the adaptivity of block-structured grids. Arbitrary grid resolutions may lead to *hanging* nodes at block boundaries which are not addressed by the previously presented discretization techniques. How a solution algorithm for partial differential equations via finite-volume methods can deal with block boundaries, is presented in this subsection.

To maintain the conservation property of finite-volume methods [**?**], block boundaries should not be interpreted as boundaries in the classical sense. Instead, conditions have to be formulated to guarantee that flows through block boundaries are conserved. The method to treat block boundaries which is used in the present thesis was presented by [**?**]. A different method was described in [**?**]. However, the first method was chosen since it allows for a fully implicit discretization of the block boundary fluxes. This method calculates fluxes through separate face segments S_l that surge from *non-matching* grid blocks. Each of these face segments gets assigned a control volume L and a control volume R, which exclusively share this face segment. An algorithm to provide the information associated with these face segments has been implemented in the present thesis. Figure 4.2 shows an example of non-matching block boundaries and the used indexing for control volumes and boundary faces. This indexing is based on a numbering of the face segments S_l.

Once the geometric data, including the information regarding the interpolation coefficients at block boundaries, has been provided, the fluxes through these boundaries can be calculated as presented in the antecedent subsections. The coefficients in the linear system corresponding to the connectivity of neighboring control volumes of different grid blocks is represented by matrix coefficients a_L^ϕ and a_R^ϕ, where $\phi \in \{u_1, u_2, u_3, p, T\}$.

4.8 Treatment of the Singularity of the Pressure-Correction Equation with Neumann Boundaries

The derived pressure-correction equation is a Poisson equation. As can be proven [**?**], the linear $N \times N$ systems surging from the presented discretization on a grid with N control volumes with Neumann boundary conditions have a kernel of dimension one, i.e.

$$\ker(A_{p'}) = \mathrm{span}(\mathbb{1}),$$

where $\mathbb{1} = (1)_{i=1,\dots,N} \in \mathbb{R}^N$ is the vector spanning the kernel. This singularity accounts for the property of incompressible flows that pressure can only be determined up to a constant. To fix this constraint, various possibilities exist [**?**]. A common method is to set the pressure-correction to zero in one reference control volume which corresponds to fixing the pressure at one reference point in the problem domain. The fixation of the pressure can be done before solving the system by applying this Dirichlet-type condition, or it can be done afterward, when pressure is calculated from the pressure-correction. This approach is not suitable for grid convergence studies since, without special interpolation, it

does not guarantee that the reference pressure-correction is taken at the correct location throughout the different grid levels.

Since some comparisons performed in the present work rely on grid convergence studies, another approach for reducing the lose constraint of the pressure-correction system is used: The reference pressure-correction is taken to be the mean value of the pressure-correction over the domain

$$p'_{\text{ref}} = \frac{\iiint_V p' \mathrm{d}V}{\iiint_V \mathrm{d}V} \approx \frac{\sum_{P=1}^{N} p'_P V_P}{\sum_{P=1}^{N} V_P}.$$

Consequently, the net pressure-correction amounts to zero. By using this fact equation (4.16) can be modified as follows

$$p_P^{(n)} = p_P^{(n-1)} + \alpha_{\mathbf{p}} \left(p'_P - p'_{\text{ref}} \right).$$

As soon as one control volume has been provided with a Dirichlet pressure boundary condition, the system is not singular anymore. For example, this is the case with fixed pressure at inlet or outlet boundary conditions, which have both been presented in section 4.7.1.

4.9 Structure of the Assembled Linear Systems

The objective of a finite-volume method is to create a set of linear algebraic equations from a discretization of partial differential equations. In the case of the discretized momentum balance, taking all contributions together, this discretization leads to the following linear algebraic equation for each control volume P

$$a_P^{u_i} u_{P_i} + \sum_{F \in NB(P)} a_F u_{F_i} = b_{P,u_i}.$$

Here, the coefficients are composed as follows

$$\begin{aligned} a_P^{u_i} &= a_P^{u_i,c} - a_P^{u_i,d} \\ a_F^{u_i} &= a_F^{u_i,c} - a_F^{u_i,d} \\ b_{P,u_i} &= b_{P,u_i}^{c} - b_{P,u_i}^{d} + b_{P,u_i}^{sc}. \end{aligned}$$

Similar expressions for the pressure-correction equation and the temperature equation can be deduced. In the case of control volumes located at boundaries some of the coefficients will be calculated in a different way. This aspect has been addressed in section 4.7.

For the decoupled iterative solution process of the Navier-Stokes equations, it is necessary to reduce the change of each dependent variable in each iteration. Normally this is done by an implicit *under-relaxation* technique, a convex combination of the solution of the linear system for the present iteration (n) and from the previous iteration $(n-1)$, with the under-relaxation parameter $\alpha_{u_i} \in (0, 1]$, where $\alpha_{u_i} = 1$ corresponds to no under-relaxation. Generally speaking, this parameter can be chosen individually for each equation. Since there are no rules for choosing these parameters in a general setting, the under-relaxation parameter for the velocities is chosen to be equal for all three velocities, $\alpha_{u_i} = \alpha_{\mathbf{u}}$ [**?**]. This choice of velocity under-relaxation has the further advantage that, in case the boundary conditions are implemented with the same intention, the linear system for each of the velocities remains unchanged except for the right-hand side. This fact helps to increase memory efficiency.

Let the solution for the linear system without under-relaxation be denoted as

$$\tilde{u}_{P_i}^{(n)} := \frac{b_{P,u_i} - \sum_{F \in NB(P)} a_F u_{F_i}}{a_P^{u_i}}.$$

This equation is only a formal expression for the case, in which the linear system for the velocity component u_i is solved directly. The described convex combination yields

$$\begin{aligned} u_{P_i}^{(n)} &:= \alpha_{\mathbf{u}} \tilde{u}_{P_i}^{(n)} + (1-\alpha_{\mathbf{u}})\, u_{P_i}^{(n-1)} \\ &= \alpha_{\mathbf{u}} \frac{b_{P,u_i} - \sum_{F \in NB(P)} a_F u_{F_i}}{a_P^{u_i}} + (1-\alpha_{\mathbf{u}})\, u_{P_i}^{(n-1)}, \end{aligned}$$

an expression that can be modified to derive a linear system whose solution is the under-relaxed velocity

$$\frac{a_P^{u_i}}{\alpha_{\mathbf{u}}} u_{i,P} + \sum_{F \in NB(P)} a_F^{u_i} u_{i,F} = b_{P,u_i} + \frac{(1-\alpha_{\mathbf{u}})\, a_P^{u_i}}{\alpha_{\mathbf{u}}}\, u_{i,P}^{(n-1)}.$$

After all matrix coefficients have been successfully calculated, the linear system can be represented by a system matrix A and a right-hand side vector b. Figure 4.3 shows the non-zero structure of a linear system for a grid consisting of a $2 \times 2 \times 2$ cell block and a $3 \times 3 \times 3$ cell block.

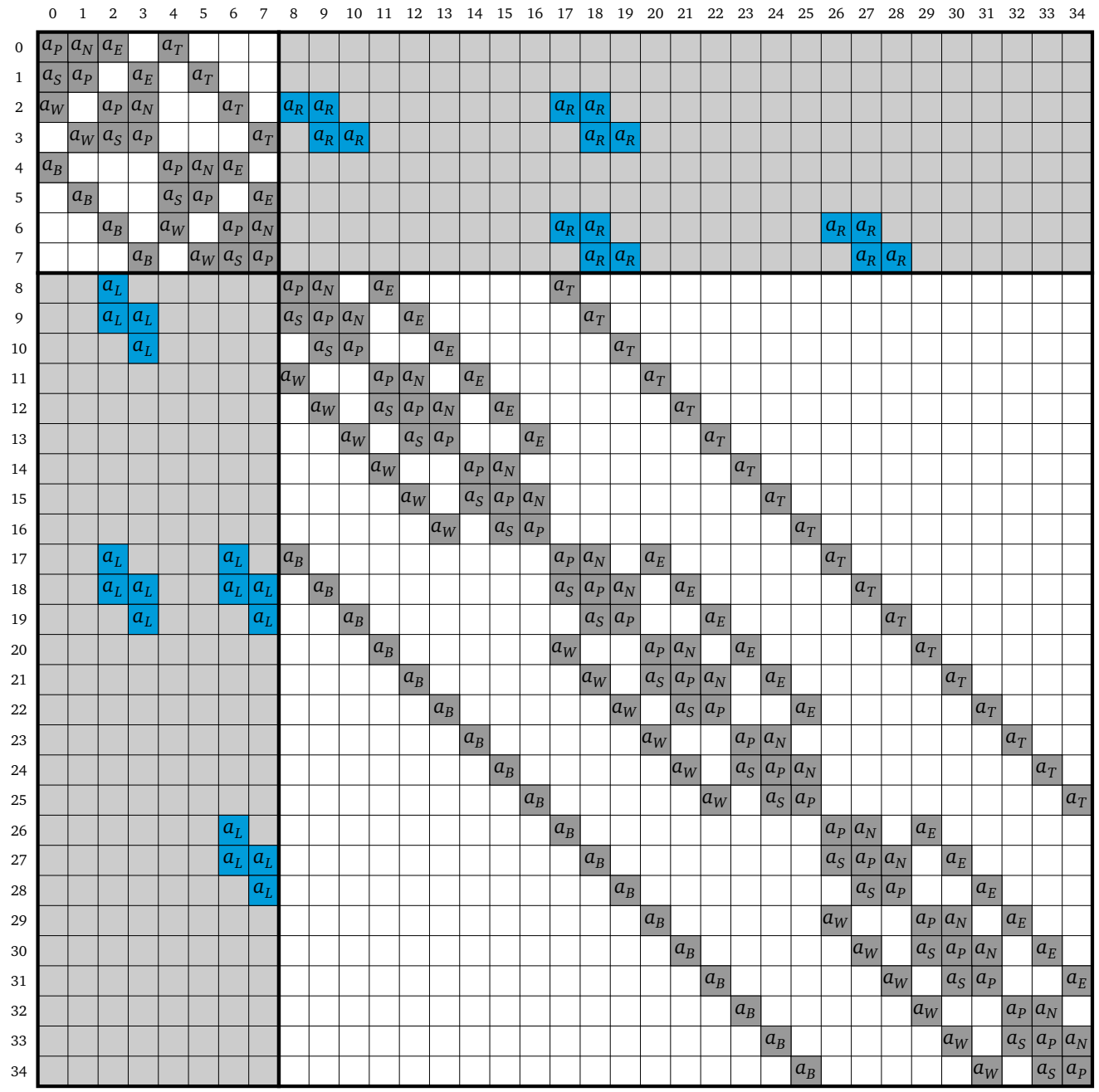


Figure 4.3: Non-zero structure of the linear systems used in the SIMPLE-algorithm for a block-structured grid consisting of one $2 \times 2 \times 2$ cell block and one $3 \times 3 \times 3$ cell block as illustrated in Figure 3.2

5 Implicit Finite-Volume Method for Incompressible Flows – Fully Coupled Approach

Since the antecedent chapter 4 already discussed the discretization details of the involved equations, this chapter aims at a comparison on the algorithmic level of the SIMPLE-algorithm, presented in section 4.3, with an implementation of a fully coupled solution algorithm. In order to maintain comparability, the discretization of the equations to be solved is not changed. Consequently, all presented differences are due to differences in the solution algorithm. Successful implementations of a fully coupled solution algorithm for incompressible Navier-Stokes equations have been presented in [**?**, **?**, **?**, **?**]. The presented work will extend the solution approach presented in [**?**] to three-dimensional domains. Furthermore, this chapter will present various approaches incorporating different degrees of velocity-to-temperature and temperature-to-velocity/pressure coupling, as they are presented in [**?**, **?**]. Finally, the structure of the resulting linear system to be solved is discussed.

5.1 The Fully Coupled Solution Algorithm – Pressure-Velocity Coupling Reconsidered

This section motivates the use of a fully solution coupled algorithm to address pressure-velocity coupling and mentions the differences to the approach presented in section 4.3. The mentioned section presented a common solution approach to solving the incompressible Navier-Stokes equations. After linearizing the equations, the momentum balances are solved, using the pressure from the previous iteration as an estimate. Generally the velocity field obtained by solving a momentum balance with a pressure estimate does not obey continuity. Hence, the velocity field and the pressure field have to be corrected. This correction in turn leads to an inferior solution with respect to the residual of the momentum balances. To avoid this iterative guess-and-correct solution process, another class of approaches dealing with the pressure-velocity coupling problematic, represented by algorithms that are *fully coupled*, will be introduced now.

The central aspect of fully coupled solution methods for Navier-Stokes equations is that, instead of solving for the velocities and pressure-corrections sequentially, the velocity field and the pressure are solved for simultaneously [**?**]. This way, every calculated velocity field obeys conservation of momentum and mass, without the need to be corrected. As a result, the under-relaxation due to the resolution of the pressure-velocity coupling is no longer required. This fact accelerates convergence significantly. The only reason for still needing an iterative solution process is the nonlinearity of the Navier-Stokes equations. This nonlinearity is accounted for by the use of the Picard iteration process, which has been presented in section 3.4. The iterations, fortunately, make room for deferred corrections as introduced in section 4.5. It is also possible to use the coupled solution algorithm to calculate pressure-corrections. Reference [**?**] uses this approach in order to solve the Navier-Stokes equations by using a SIMPLE-type method as preconditioner.

On the downside, implementations of fully coupled solution methods require significantly more system memory than segregated methods. Since coupled solution algorithms simultaneously solve for a higher number of unknowns, a higher amount of information has to be available at all times, including the higher storage requirements for the associated linear system. Furthermore, the bad condition of the linear algebraic system [**?**] tends to slow down the convergence of the equation solver algorithm.

As in the case of segregated methods, it is not advisable to solve the mass balance equation directly [**?**]. Instead of deriving an equation for the pressure-correction p', as shown in section 4.3, an equation for the pressure itself is derived, using the pressure-weighted interpolation method which has been introduced in section 4.2. More specifically equation (4.5) is adapted for the use in a fully coupled algorithm

$$u_{i,f}^{(n)} = \left[\left(1-\gamma_f\right)u_{i,P}^{(n)} + \gamma_f u_{i,Q}^{(n)}\right] - \left(\left(1-\gamma_f\right)\frac{V_P}{a_P^{u_i}} + \gamma_f \frac{V_Q}{a_Q^{u_i}}\right)\left[\underline{\left(\frac{\partial p}{\partial x_i}\right)_f^{(n)}} - \frac{1}{2}\left(\left(\frac{\partial p}{\partial x_i}\right)_P^{(n-1)} + \left(\frac{\partial p}{\partial x_i}\right)_Q^{(n-1)}\right)\right].$$

The adaption comprises the removal of the last term of equation (4.5) accounting for the under-relaxation since, according to [**?**], no under-relaxation is needed in fully coupled algorithms. This fact is equivalent to an under-relaxation factor

$\alpha_{\mathbf{u}} = 1$. Furthermore, the underlined partial derivatives are going to be treated implicitly. This treatment leads to the semi-implicit pressure equation

$$\sum_{F\in NB(P)} \rho\left[\left(1-\gamma_f\right)u_{i,P}^{(n)} + \gamma_f u_{i,F}^{(n)}\right] S_f - \rho\left(\left(1-\gamma_f\right)\frac{V_P}{a_P^{u_i}} + \gamma_f \frac{V_F}{a_F^{u_i}}\right)\left[\left(\frac{\partial p}{\partial x_i}\right)_f^{(n)}\right] S_f$$
$$= -\sum_{F\in NB(P)} \rho\left(\left(1-\gamma_f\right)\frac{V_P}{a_P^{u_i}} + \gamma_f \frac{V_F}{a_F^{u_i}}\right)\left[\frac{1}{2}\left(\left(\frac{\partial p}{\partial x_i}\right)_P^{(n-1)} + \left(\frac{\partial p}{\partial x_i}\right)_F^{(n-1)}\right)\right] S_f .$$

Even though this equation is solved for the pressure p, while equation (4.14) is solved for the pressure-correction p', the matrix coefficients from the discretization of the pressure-correction equation (4.19) and the calculation of the right-hand side given in equation (4.15), remain the same. For the implicit coupling to the velocities u_i, also denoted as the *pressure-to-velocity* coupling, additional matrix coefficients have to be considered. These coefficients can be calculated as follows

$$a_F^{p,u_i} = \rho\,\gamma_f S_f \quad \text{and} \quad a_P^{p,u_i} = \sum_{F\in NB(P)} \rho\,(1-\gamma_f) S_f .$$

On the other side, the discretized momentum balance yields matrix coefficients to account for the implicit *velocity-to-pressure* coupling. These coefficients are calculated as follows

$$a_F^{u_i,p} = \gamma_f S_f \quad \text{and} \quad a_P^{u_i,p} = \sum_{F\in NB(P)} (1-\gamma_f) S_f .$$

If the Boussinesq approximation is used implicitly, an additional coefficient $a_P^{u_i,T}$ to account for the velocity-to-temperature coupling has to be considered. Section 5.2 will present different methods of temperature coupling in detail. Algorithm 2 shows the main steps of the fully coupled solution algorithm.

Algorithm 2 Fully Coupled Solution Algorithm

```
INITIALIZE variables
while (convergence criterion not accomplished) do
    if (temperature coupling) then
        SOLVE the linear system for velocities, pressure, and temperature
    else
        SOLVE the linear system for velocities and pressure
    end if
    CALCULATE mass fluxes using (4.5)
    if (coupled scalar equation) then
        SOLVE scalar equation as described in section 4.6
    end if
end while
if (decoupled scalar equation) then
    SOLVE scalar equation as described in section 4.6
end if
```

5.2 Coupling to the Temperature Equation

The present work also aims at analyzing the efficiency of different methods to couple the temperature equation to the velocities and vice versa. Consequently, this section discusses different approaches of handling the coupling of the temperature equation to the Navier-Stokes equations if the Boussinesq approximation, as introduced in section 2.5.2, is used. The effectiveness of realizing a strong coupling of the temperature equation to the Navier-Stokes equations depends on the flow problem. The physical coupling tends to increase in flow scenarios of natural convection, where the temperature difference, and hence the buoyancy term in the Navier-Stokes equations, dominates the fluid movement [**?**, **?**]. It is supposed that, the higher the physical coupling of temperature and flow, the greater the benefits of using an implementation that uses a strong coupling to the temperature equation. The following subsections present different approaches to achieving implicit inter-variable coupling between velocities, pressure, and temperature.

5.2.1 Decoupled Approach – Explicit Velocity-to-Temperature Coupling

The decoupled approach for addressing velocity-to-temperature coupling is similar to the treatment of the velocity-to-temperature coupling described in section 4.5. If this approach is chosen, no special measures have to be taken. The momentum balances receive a contribution $b_P^{u_i,T}$ with

$$b_P^{u_i,T} := -\rho\beta\left(T_P^{(n-1)} - T_0\right) g_i V_P,$$

that handles the coupling explicitly by using the temperature result of the previous outer iteration. In some cases, it might be necessary to under-relax the temperature each iteration [**?**].

5.2.2 Implicit Velocity-to-Temperature Coupling

It is possible to realize the coupling described in the previous subsection by implicitly considering the temperature in the momentum balances. This approach leads to an additional matrix coefficient $a_P^{u_i,T}$ and an additional contribution to the right-hand side, accounting for the coupling to the temperature equation,

$$a_P^{u_i,T} = \rho\beta g_i V_P \quad \text{and} \quad b_P^{u_i,T} = \rho\beta T_0 g_i V_P.$$

5.2.3 Temperature-to-Velocity/Pressure Coupling – Newton-Raphson Linearization

The previous subsections discussed the velocity-to-temperature coupling and so far, only the momentum balances were affected by the realization of the coupling methods. Independently, it is possible to couple the temperature equation, not only to the momentum balances, but also to the pressure equation. The purpose of this subsection is to present a discretization that achieves this opposite type of coupling, which will be referred to as *temperature-to-velocity/pressure* coupling.

In section 4.6, the nonlinear partial differential equations were linearized using a Picard iteration method. Specifically the convective term $\rho u_j T$ was linearized by taking the mass flux from the antecedent solve of the momentum and pressure-correction equations. If the decoupled approach is used, this treatment remains valid. However, if an implicit temperature coupling is sought, a coupling of the temperature equation to the momentum balances which considers the mass fluxes is desirable. Methods that exhibit the described property can be found in [**?**, **?**, **?**, **?**]. A common denomination of the therein used linearization method is called *Newton-Raphson linearization*. This method can be interpreted as a bilinear approximation of the convective term by the linear terms of the respective Taylor polynomial, similar to Newton-like methods to solve nonlinear equations [**?**]. Newton-like methods are known for their faster, up to quadratic convergence compared to methods with linear convergence, like the Picard iteration. However, the intention of using a Newton-like linearization for the convective term of the temperature equation is different. Here, the Newton-Raphson linearisation is utilized to enhance the inter-variable coupling in order to accelerate convergence. For a Newton-like linearization to accelerate convergence, instead of only the convective term, the whole equation should use a Newton method for the nonlinear solution algorithm. As a result, the *Newton-Raphson linearization* is not commonly used within segregated, SIMPLE-like solution algorithms [**?**].

The Newton-Raphson linearization technique is applied to the convective term of the temperature equation as follows. A first order Taylor approximation of the mass-specific convective flux through the boundary face S_f around the $(n-1)$st iteration value yields for the approximation at the nth iteration

$$\begin{aligned}
\left(u_{j,f}\, T_f\right)^{(n)} &\approx \left(u_{j,f}\, T_f\right)^{(n-1)} + \frac{\partial}{\partial u_{j,f}}\left(u_{j,f}\, T_f\right)^{(n-1)}\left(u_{j,f}^{(n)} - u_{j,f}^{(n-1)}\right) + \frac{\partial}{\partial T_f}\left(u_{j,f}\, T_f\right)^{(n-1)}\left(T_f^{(n)} - T_f^{(n-1)}\right) \\
&= (u_{j,f}\, T_f)^{(n-1)} + {T_f}^{(n-1)}({u_{j,f}}^{(n)} - {u_{j,f}}^{(n-1)}) + {u_{j,f}}^{(n-1)}({T_f}^{(n)} - {T_f}^{(n-1)}) \\
&= \underline{{u_{j,f}}^{(n-1)}{T_f}^{(n)}} + {u_{j,f}}^{(n)}{T_f}^{(n-1)} - {u_{j,f}}^{(n-1)}{T_f}^{(n-1)}.
\end{aligned} \tag{5.1}$$

A comparison with section 4.5.1 shows that the underlined term coincides with the term from the usual linearization. The first of the two new terms will be treated implicitly and explicitly after using the pressure-weighted interpolation method from section 4.2 to interpolate the value of $u_{j,f}$. The second term will be treated explicitly. The use of the pressure-weighted interpolation method does create not only a temperature-to-velocity but also a temperature-to-

pressure coupling. Applying the pressure-weighted interpolation, equation (5.1) for the mass-specific convective flux reads

$$\begin{aligned}\left(u_{j,f}\,T_f\right)^{(n)} \approx\; & {u_{j,f}}^{(n-1)}\,{T_f}^{(n)} \\ & + {T_f}^{(n-1)}\left[\left(1-\gamma_f\right)u_{j,P}^{(n)} + \gamma_f u_{j,Q}^{(n)}\right] \\ & - {T_f}^{(n-1)}\left(\left(1-\gamma_f\right)\frac{\alpha_{\mathbf{u}}V_P}{a_P^{u_j}} + \gamma_f\frac{\alpha_{\mathbf{u}}V_Q}{a_Q^{u_j}}\right)\left[\left(\frac{\partial p}{\partial x_j}\right)_f^{(n)} - \frac{1}{2}\left(\frac{\partial p}{\partial x_j}\right)_P^{(n-1)} - \frac{1}{2}\left(\frac{\partial p}{\partial x_j}\right)_Q^{(n-1)}\right] \\ & - {u_{j,f}}^{(n-1)}\,{T_f}^{(n-1)}.\end{aligned}$$

If this semi-implicit, semi-discrete equation is discretized completely, the new matrix and right-hand side contributions can be calculated to

$$a_F^{T,u_i} = \rho\, T_f^{(n-1)}\gamma_f\, n_{f,i} S_f$$

$$a_P^{T,u_i} = \sum_{F\in NB(P)} \rho\, T_f^{(n-1)}\left(1-\gamma_f\right) n_{f,i} S_f$$

$$a_F^{T,p} = -\rho\, T_f^{(n-1)}\left(\left(1-\gamma_f\right)\frac{\alpha_{\mathbf{u}}V_P}{a_P^{u_i}} + \gamma_f\frac{\alpha_{\mathbf{u}}V_F}{a_F^{u_i}}\right)\frac{S_f}{(\mathbf{x}_P-\mathbf{x}_F)\cdot\mathbf{n}_f} = \rho\, T_f^{(n-1)} a_F^p$$

$$\text{and}\quad a_P^{T,p} = -\sum_{F\in NB(P)} a_F^{T,p}.$$

The explicit parts resulting from the Newton-Raphson linearization yield additional contributions to the right-hand side

$$b_P^{T,\mathrm{NR}} = \rho\, u_{j,f}^{(n-1)} T_f^{(n-1)} n_{f,j} S_f - \rho\, T_f^{(n-1)}\left(\left(1-\gamma_f\right)\frac{\alpha_{\mathbf{u}}V_P}{a_P^{u_j}} + \gamma_f\frac{\alpha_{\mathbf{u}}V_Q}{a_Q^{u_j}}\right)\left[\frac{1}{2}\left(\frac{\partial p}{\partial x_j}\right)_P^{(n-1)} + \frac{1}{2}\left(\frac{\partial p}{\partial x_j}\right)_Q^{(n-1)}\right] n_{j,f} S_f.$$

5.3 Boundary Conditions on Domain and Block Boundaries

The treatment of boundary conditions and transitional conditions at block boundaries is very similar to the way presented in section 4.7. The main difference results from the fact that, in addition to the necessary changes from section 4.7, modifications accounting for the new matrix coefficients from the discretized momentum balance and the pressure equation are necessary. The purpose of this section is, therefore, to highlight these additional changes.

How Dirichlet boundary conditions, also known as inlet boundaries, affect the matrix coefficients that only correspond to the velocities has been presented in section 4.7.1. It is common to use Neumann boundary conditions for the pressure at Dirichlet boundaries with respect to the velocities [**?**]. Under the assumption that the gradient of the pressure vanishes at inlet boundaries, the matrix coefficients for the velocity-to-pressure coupling have to be modified to

$$a_P^{u_i,p} = a_P^{u_i,p} + n_{f,i} S_f.$$

Wall boundary conditions are treated likewise with the additional modifications to the right-hand side contribution as presented in section 4.7.2.

The treatment of velocity Dirichlet boundary conditions within the pressure equation is straightforward since they correspond to mass fluxes through the boundary faces. Hence, the matrix coefficients remain unchanged, and only the right-hand side contributions embrace the fact that no implicit interpolation is necessary to calculate the mass flux through the boundary face.

In the case of Dirichlet pressure boundary conditions, an additional contribution to the right-hand side of the momentum balances has to be considered

$$b_P^{u_i} = b_P^{u_i} + p_f\, n_{i,f} S_f.$$

As presented in section 4.7 a *lagged* Dirichlet boundary condition is used for the velocities in the discretized momentum balances. Different to the presentation in section 4.7 the boundary mass fluxes in the pressure equation are considered implicitly. As a result, the boundary treatment has to consider contributions to the coefficients accounting for the pressure-to-velocity coupling in the pressure equation

$$a_P^{p,u_i} = a_P^{p,u_i} + \rho n_{f,i} S_f .$$

The coefficients for the pressure are modified as usual for Dirichlet boundary conditions, which was shown in section 4.7.

Since at Dirichlet boundaries the values of the velocities are known exactly, the Newton-Raphson linearization technique does not apply to the velocities on boundary faces. As a result, the linearization of the convective term of the temperature equation becomes unnecessary.

At block boundaries the new matrix coefficients $a_L^{u_i,p}$ and $a_R^{u_i,p}$ for the velocity-to-pressure coupling and the matrix coefficients a_L^{p,u_i} and a_R^{p,u_i} have to be taken into account. This fact does not present any drawbacks to the presented approach. If additionally temperature-to-velocity/pressure coupling is applied, corresponding matrix coefficients as for the coupling have to be considered. The implicit Boussinesq approximation does not require additional matrix coefficients at block boundaries as this is a local approximation.

5.4 Assembly of Linear Systems – Final Form of Equations

Concluding this chapter, the final form of the derived linear algebraic equations is presented. Based on the presented discretization, each equation only contains variable values of neighboring control volumes. The set of five equations reads

$$a_P^{u_i} u_{P,i} + \sum_{F \in NB(P)} a_F^{u_i} u_{F,i} + \underbrace{a_P^{u_i,p} p_P + \sum_{F \in NB(P)} a_F^{u_i,p} p_F}_{\text{Pressure-velocity coupling}} + \underbrace{a_P^{u_i,T} T_P}_{\text{Boussinesq approximation}} = b_{P,u_i} \quad i = 1, \ldots, 3$$

$$a_P^{p} p_P + \sum_{F \in NB(P)} a_F^{p} p_F + \underbrace{\sum_{j=1}^{3} \left(a_P^{p,u_j} u_{P,j} + \sum_{F \in NB(P)} a_F^{p,u_j} u_{F,j} \right)}_{\text{Pressure-velocity coupling}} = b_{P,p}$$

$$a_P^{T} T_P + \sum_{F \in NB(P)} a_F^{T} T_F + \underbrace{\sum_{j=1}^{3} \left(a_P^{T,u_j} u_{P,j} + \sum_{F \in NB(P)} a_F^{T,u_j} u_{F,j} \right) + a_P^{T,p} p_P + \sum_{F \in NB(P)} a_F^{T,p} p_F}_{\text{Newton-Raphson linearization}} = b_{P,T} .$$

The different degrees of coupling between the different equations will affect the matrix structure. Depending on the variable arrangement, other matrix structures are possible. Figure 5.1 shows the resulting block matrix structure if the fields are treated as interlaced quantities. This variable arrangement corresponds to a linear subsystem for each control volume:

$$\begin{bmatrix} a_P^{u_1} & 0 & 0 & a_P^{u_1,p} & a_P^{u_1,T} \\ 0 & a_P^{u_2} & 0 & a_P^{u_2,p} & a_P^{u_2,T} \\ 0 & 0 & a_P^{u_1} & a_P^{u_3,p} & a_P^{u_3,T} \\ a_P^{p,u_1} & a_P^{p,u_2} & a_P^{p,u_3} & a_P^{p} & 0 \\ a_P^{T,u_1} & a_P^{T,u_2} & a_P^{T,u_3} & a_P^{T,p} & a_P^{T} \end{bmatrix} \begin{bmatrix} u_{1,P} \\ u_{2,P} \\ u_{3,P} \\ p_P \\ T_P \end{bmatrix} + \sum_{F \in NB(P)} \begin{bmatrix} a_F^{u_1} & 0 & 0 & a_F^{u_1,p} & a_F^{u_1,T} \\ 0 & a_F^{u_2} & 0 & a_F^{u_2,p} & a_F^{u_2,T} \\ 0 & 0 & a_F^{u_1} & a_F^{u_3,p} & a_F^{u_3,T} \\ a_F^{p,u_1} & a_F^{p,u_2} & a_F^{p,u_3} & a_F^{p} & 0 \\ a_F^{T,u_1} & a_F^{T,u_2} & a_F^{T,u_3} & a_F^{T,p} & a_F^{T} \end{bmatrix} \begin{bmatrix} u_{1,F} \\ u_{2,F} \\ u_{3,F} \\ p_F \\ T_F \end{bmatrix} = \begin{bmatrix} b_P^{u_1} \\ b_P^{u_2} \\ b_P^{u_3} \\ b_P^{p} \\ b_P^{T} \end{bmatrix} .$$

Figure 5.2 shows the non-zero structure of a block matrix for the same case if field interlacing is used. The variable arrangement only affects the storage of the matrices, not the way the matrix is used, since it is possible to reorder the matrix entries arbitrarily through the underlying index sets and obtain other matrix representations. One commonly used matrix representation is shown in Figure 5.3.

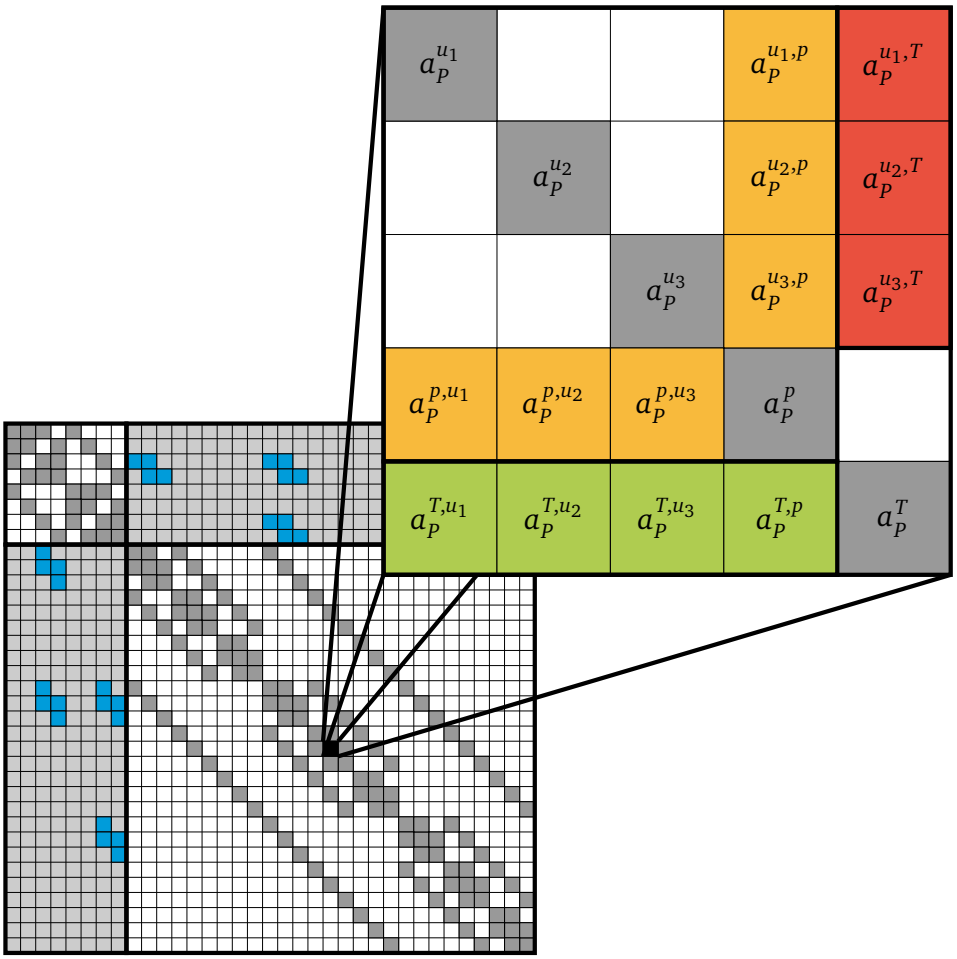


Figure 5.1: Non-zero structure of block submatrices of the linear systems used in the coupled solution algorithm for a block-structured grid consisting of one $2 \times 2 \times 2$ cell block and one $3 \times 3 \times 3$ cell block. The yellow coefficients account for the pressure-velocity coupling, the red coefficients correspond to the velocity-to-temperature coupling due to the implicit Boussinesq approximation and the green coefficients result from the Newton-Raphson linearization technique and address the temperature-to-velocity/pressure coupling.

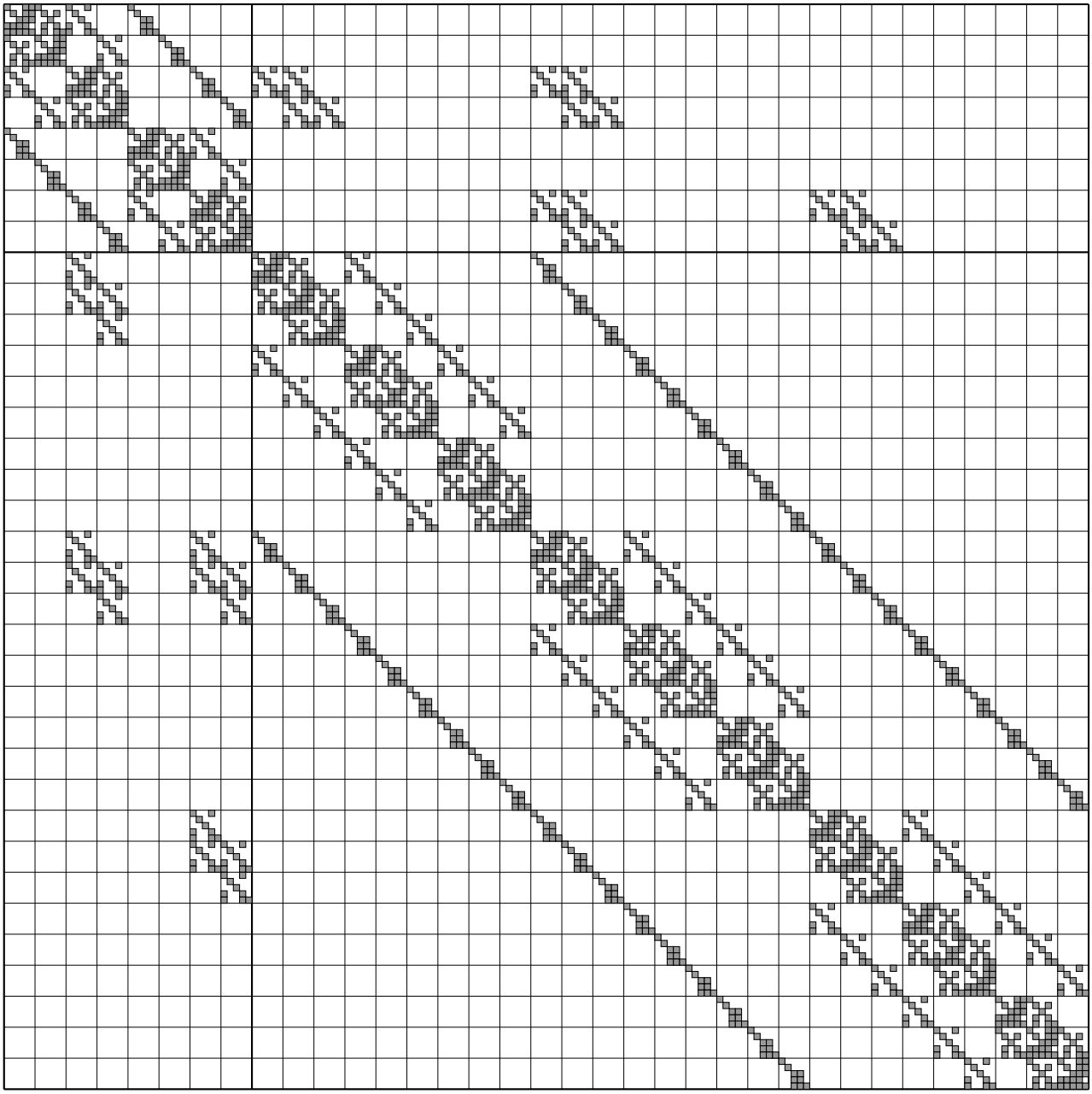


Figure 5.2: Non-zero structure of the linear system used in the coupled solution algorithm for a block-structured grid consisting of one $2 \times 2 \times 2$ cell block and one $3 \times 3 \times 3$ cell block. The variables have been interlaced, and the matrix consists of blocks as shown in Figure 5.1.

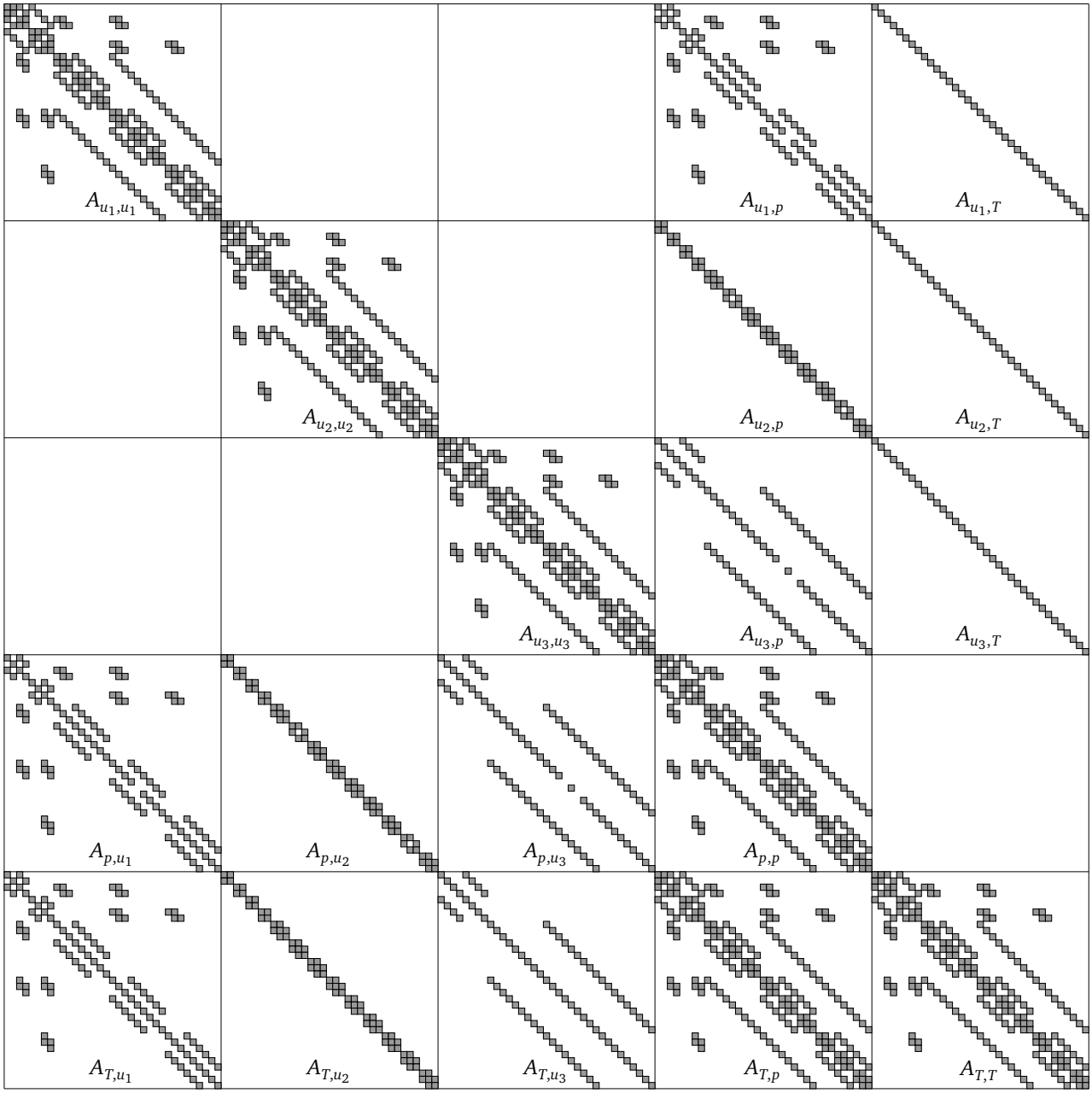


Figure 5.3: Non-zero structure of the linear system used in the coupled solution algorithm for a block-structured grid consisting of one $2 \times 2 \times 2$ cell block and one $3 \times 3 \times 3$ cell block. The variables have been ordered in a way that each major matrix block refers to only one variable or the coupling between exactly two variables as it is denoted by the corresponding subindices.

6 CAFFA Framework

This chapter presents the CAFFA framework that has been extended for the present thesis. *CAFFA* is an acronym, which stands for *Computer Aided Fluid Flow Analysis*. The solver framework is based on Perić's CAFFA code [**?**] which has been modified to handle three-dimensional problem domains and block-structured grids with non-matching blocks. Furthermore, the framework has been parallelized making extensive use of the PETSc library. One characteristic of the solver framework is that arbitrary block boundaries are handled in a fully implicit way. Details on how those transitional conditions are handled can be found in section 4.7.3. The framework consists of two different solver algorithms, a segregated solver and a fully coupled solver, whose theoretical basis has been introduced in chapters 4 and 5 respectively. In addition, the framework features an own implementation of a grid generator for block-structured grids as well as a mapper program to convert grids generated by a commercial meshing tool into a format understandable by the CAFFA code. To prepare the grid data for later use in the solvers, a preprocessor program has been integrated into the framework. Furthermore, the solver programs exhibit different export functionalities not only to visualize the numerical grid and the results within ParaView [**?**] but also to export vectors and matrices to MATLAB ® [**?**].

The following sections will now briefly introduce the PETSc framework and present the components of the CAFFA framework.

6.1 PETSc Framework

PETSc is an acronym for *Portable Extensible Toolkit for Scientific Computation* and is a software suite which comprises data structures and an extensive set of linear solver routines [**?**,**?**]. A great part of the data structures is designed to be used for parallelized applications on high-performance computers. Also, the solver routines have been parallelized using the MPI standard for message passing. PETSc provides tools which can be used to build large-scale application codes for scientific calculations. The great advantage of PETSc is that it comprises multiple additional layers of abstraction. This abstraction makes the use of PETSc in applications straightforward and allows the users to focus on the essential implementations of their own code instead of having to deal with parallel data structures and algorithms, since this represents an additional source of programming errors. PETSc provides different interfaces through which the user can parallelize his code, a fact that furthermore increases readability and maintainability of the developed code. Last but not least, the implementations used for data structures and solvers in PETSc are not only verified and updated on a regular basis, they have furthermore proven to be highly efficient in the creation of large-scale computer applications on high-performance clusters [**?**, **?**, **?**].

The data structures of PETSc reflect the commonly used elements in scientific codes. Vectors, matrices, and index sets are among these basic data structures. Beyond that, PETSc offers a variety of options for its data structures. Vectors can either be used sequentially or in parallel, in which case PETSc distributes the vector components to the involved processes inside a specified MPI communicator. Furthermore, parallel matrix formats are available that allow the processes to assemble different parts of one global matrix simultaneously. PETSc contains other data structures which can be used to fetch and distribute either vector or matrix elements from remote processes.

In addition to parallel data structures, PETSc contains a large collection of parallel solvers for linear systems. The major part comprises Krylov subspace methods. An overview over classical Krylov subspace methods can be found in [**?**] Among the variety of solver options that are available for further optimization of the performance, the Krylov subspace methods can be combined with elements of another thorough collection of preconditioners and other third party packages like Hypre [**?**] or ML [**?**]. For further details on the available data structures and subroutines, the reader is referred to [**?**, **?**].

In addition to the mentioned tools, PETSc also comes with an internal profiler tool which collects a variety of information on the used PETSc objects and bundles them into a human readable log file. The log file and other dynamic output, which can be triggered by command line arguments, facilitate the identification of performance bottlenecks and accelerate the optimization process of developed applications.

6.2 Grid Generation and Preprocessing

A grid generator has been developed to generate hexahedral block-structured grids. In order to provide flexibility for testing locally refined and skewed grids, the grid generator deploys a random number generator which moves grid points within the domain and varies the number of grid cells of each block optionally. Key feature of the developed framework is the handling of block-structured locally refined grids with non-matching block interfaces. The neighboring relations are represented by a special type of boundary condition.

After the grid has been generated, a preprocessor program comes to application which prepares the numerical grid for the use in the solver program. An essential part of the preprocessing step is the application of various matching algorithms depending on the type of block boundary condition. The preprocessor program differentiates between three block boundary types of which each is a subtype of the other. Figure 6.1 illustrates these different types of block boundaries and shows the resulting relation between block boundary cells. The most simple block boundary condition is given by a *one-to-one* relation between two neighboring boundary cells. The generalization of this condition involves several boundary cells that share only one neighboring boundary cell in the neighboring block. This relation is also known as a *many-to-one* relation. The third boundary condition on block boundaries implements the most general *many-to-many* relation, which needs special consideration in the preprocessing step, because the number of neighbors one cell can have is not known a priori. What makes the third type of boundary conditions more difficult than the first two, is the geometry of the resulting boundary faces. The current implementation uses a polygon clipper to calculate the polygons resulting from the intersection of two boundary faces of neighboring blocks. Based on the polygon of the intersection, all further geometric data like interpolation factors and boundary face area can be calculated for the later use within the solver application.

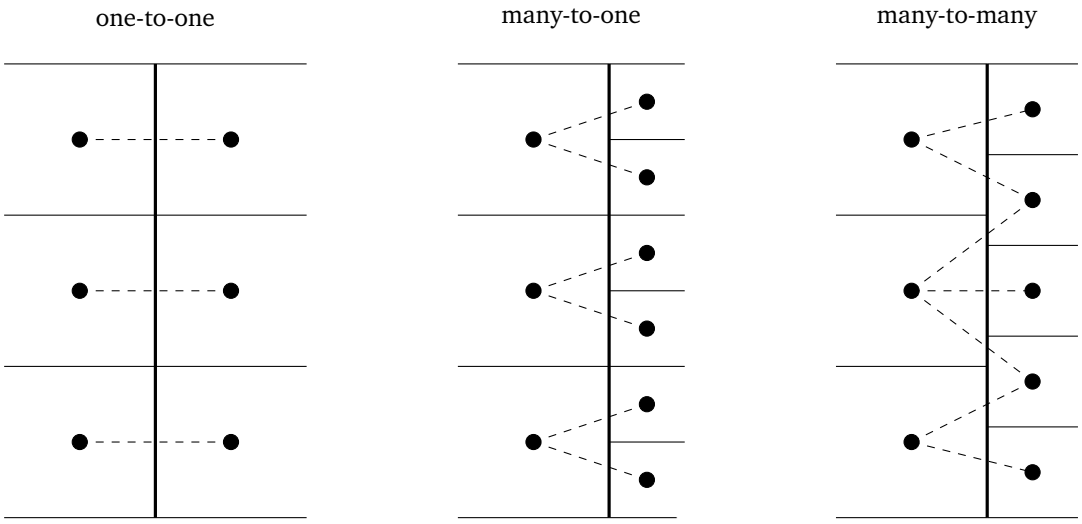


Figure 6.1: Different types of block boundaries for a two-dimensional block-structured grid

6.3 Implementation of CAFFA

This section provides an overview of certain implementation aspects of the developed CAFFA framework. Since applications for modern computers have to be able to use the provided resources efficiently, the concept for the parallelization of the application is presented. A separate section presents, how convergence is monitored and controlled. Later on, this section shows how the PETSc data structures that store the variables are used to realize the data composition as a central part of the parallelization process.

6.3.1 The Message-Passing Model

The PETSc framework makes use of the message-passing computational model [**?**]. This model assumes that a set of processes with local memory is able to communicate with other processes by *passing*, i.e. sending and receiving, messages. *MPI* is an acronym for *Message Passing Interface* and is a software library that collects features of message-passing systems. More importantly, MPI provides a widely used standard that assures the portability of applications that make use of this library, as does PETSc.

The message-passing model is not only reflected in the application as a programming paradigm, but also in the design of parallel computers and their communication network. From this perspective, the message-passing model is an analog to the distributed memory design which encapsulates the idea of local memory and the exchange of data through a network. High-performance computers are seldom designed exclusively as distributed memory systems. They rather additionally incorporate features of another model: the shared memory model. This memory model is characterized by a common address space which each of the processes can access uniformly, i.e. with a location independent latency, or non-uniformly. Latter systems are also called *NUMA* (*Non-Uniform Memory Access*) systems. A widely used system design,

as used in the system on which the performance tests of chapter **??** are carried out, comprises features of both models by using NUMA computing nodes that share local memory and are able to exchange data with other NUMA nodes via an interconnect and the passing of messages.

One important advantage of using software that uses the message passing model and hence applications that were parallelized within this programming paradigm is their compatibility with computers that use combinations of distributed and shared memory.

6.3.2 Convergence Control

One part of the PETSc philosophy is to provide maximum flexibility in choosing solvers and parameters from within the command line. However, the developed framework implements a nonlinear iteration process without using the *SNES* (*Scalable Nonlinear Equations Solvers*) objects provided by PETSc. This fact creates the need for a hard-coded implementation of convergence control. Convergence control for the used Picard iteration method comprises two parts. One part controls the overall convergence and determines when the calculations have finished. The other part controls the convergence on an outer iteration basis. The convergence criteria are met if the actual residual falls below the upper bound for the residual r_{final}. The bound for final convergence is determined to

$$r_{final} = r_{initial} * 10^{-8},$$

which states that a relative decrease in the initial residual of 10^{-8} indicates convergence. This criterion will be fixed for all further analyses since it controls the overall accuracy of the solver results and hence maintains comparability of solver performance, unless otherwise stated.

The other convergence criterion can be handled with more flexibility since it controls the relative decrease of the residual in each outer iteration. This parameter should not affect the overall accuracy but instead convergence speed, within the individual bounds on the solver algorithm. Low relative solver tolerances will not always benefit convergence speed. While high relative solver tolerances will decrease the number of needed inner iterations, the number of outer iterations might increase nullifying the benefit of faster convergence of the linear solver. The convergence criterion for each outer iteration is implemented as

$$r^{(k)}_{final} = r^{(k)}_{initial} * \mathrm{rtol},$$

where rtol indicates the relative decrease of the residual in each outer iteration.

6.3.3 Indexing of Variables and Treatment of Boundary Values

All variables needed by the CAFFA solver are represented by one-dimensional arrays. To establish a mapping between a location (i, j, k) of the numerical grid and the corresponding location (ijk) inside a variable's array the following formula is used

$$(ijk) = N_i N_j (k-1) + N_j (i-1) + j,$$

where N_i and N_j are the numbers of grid cells in the respective coordinate direction plus two additional boundary cells. The inclusion of the boundary cells circumvents the need to provide an additional data structure for boundary values. Furthermore, the same mapping rule can be used for the grid vertex coordinates. This indexing of variables will be used to assemble the matrices surging from the discretization process and lead to rows for the boundary values that are decoupled from the rest of the linear system. PETSc provides two different approaches to handle boundary values. In the first approach another abstraction layer is introduced via an object which redistributes the data and removes the rows containing the boundary values. This approach has been tested with the conclusion that the redistribution causes not only significant overhead but also hides important data for debugging solver convergence. In the present work, a second approach is used which retains boundary values and adapts the right-hand side accordingly. This approach has proven to be more efficient with the drawback of memory overhead and the need to reset the boundary values after a linear system has been solved with a high relative tolerance.

The presented indexing model is not capable of handling block-structured grids or grids that have been distributed across various processes within the MPI programming model. This property is implemented by additional mappings that return a constant offset $(ijk)_b$ for each block and $(ijk)_p$ for each process

$$(ijk) = (ijk)_p + (ijk)_b + N_i N_j (k-1) + N_j (i-1) + j.$$

The introduction of the process offset is needed to provide a mapping between locally and globally indexed values. Within one process only the local indexing

$$(ijk)_{loc} = (ijk)_b + N_i N_j (k-1) + N_j (i-1) + j$$

will be used since these indexes can be mapped directly to memory addresses. For inter-process communication, however, global indices have to be provided to the respective PETSc subroutines by adding the process offset

$$(ijk)_{glo} = (ijk)_{loc} + (ijk)_p. \tag{6.1}$$

It should be noted that the presented mappings do not include the index mappings from FORTRAN applications, which use 1-based indexing, to the PETSc subroutines, which use 0-based indices.

As presented in chapter 5, the use of a coupling algorithm leads to a global linear system that contains the linear algebraic equations for different variables. The implementation used for the present thesis interlaces these variables, a fact that increases the locality of the data and hence improves memory efficiency. The presented mapping is easily extended to handle arrays of values that contain n_{eq} variables and use an interlaced storing approach by multiplying (6.1) with the number of interlaced values

$$(ijk)_{interglo} = n_{eq} * \big((ijk)_{glo} - 1\big) + c_{eq},$$

where $c_{eq} = 1, \ldots, n_{eq}$.

6.3.4 Domain Decomposition and the Exchange of Ghost Values

The developed CAFFA framework is able to solve a given problem across different processes. For this, before each solve, the relevant data has to be distributed among all involved processes. Within the solver parallelization, three types of this data distribution are considered with respect to their frequency of communication and redistribution among all or only a part of the processes involved in the solution procedure.

The first type of data refers to the distribution of stationary data that will not change throughout the solution process. This type of data refers mostly to problem geometry-related data, as for example the coordinates of the grid points. The distribution of the data is already part of the domain decomposition process and can be done before the solver application starts, as it is implemented in the developed solver framework. In the case, in which local dynamic refinement strategies are used, the geometric data at block boundaries will have to be redistributed. If additionally dynamic load balancing is used, even geometric data from inside the blocks of the problem domain might have to be redistributed. This use case, however, imposes other design guidelines for the involved processes which lay outside of the scope of the present thesis.

The distribution of stationary data takes place throughout the preprocessing program within the developed framework before the CAFFA solver is launched. Each processor is assigned a binary file with the respective data. Since the solver is not able to handle adaptive grid refinement or dynamic load balancing, this approach is straightforward.

The second type of data has to be distributed regularly since this data changes at least every outer iteration. The necessity to interchange values, also known as *ghosting*, between processors surges when calculations of variable gradients or coefficients are made for block boundary control volumes. PETSc provides special interfaces for ghosted vectors that allow to perform different types of vector updates through top-level subroutines. The central part are the involved vector scatter and gather routines which allow not only to calculate contributions to source terms of control volumes located at block boundaries but also to update ghost values for the control volumes that rely on variable values from neighboring blocks to calculate fluxes or gradients. The present work developed an approach for ghosting values which is not symmetric in the sense that each block saves ghost values of the neighboring blocks. On the contrary, for each block boundary only one of the neighbored blocks is responsible for the calculation of the fluxes and gradient contributions. This convention maintains the same amount of data communicated through the interconnect while splitting the computational effort almost in half.

Another advantage of the PETSc environment for ghosting values that reside in the memory space of other processes is the unified approach of data coherence since, on the top level, only one vector object to store all data related to a variable, e.g. the velocity u_1, has to be created. Through precise information provided in the creation step of these vector objects, PETSc distributes the vector object to all involved processes and uses the provided information to manage the ghosting through the PETSc internal vector scatter and gather routines.

The present solver framework treats the calculation of matrix coefficients for block boundary control volumes in a special way. To reduce communication overhead and redundant data, only one of the two involved processes calculates these coefficients and then sends them to the respective neighbor when the matrix is being assembled. The data needed to calculate the matrix coefficients of a neighboring process embraces not only the values of the dominant variables

from the last outer iteration but also the values of the cell center gradients. For the ghosting of variable values, PETSc offers special vector types which comprise a user-friendly interface to use the gather and scatter routines provided by PETSc. During the creation of these vectors, a set of global indices referring to the values that are to be ghosted, has to be provided. After this, the interchange of values is realized through a single subroutine call to VecGhostUpdate since PETSc handles the communication process which is necessary to accomplish the passing of values. Hence, the local representation of a ghosted vector, not only comprises the data local to one processor, but also provides space for ghosted values that are updated repeatedly. The parallel vector layout and the process of ghosting are schematically shown in Figure 6.2 for a two-dimensional domain consisting of two grid blocks distributed among two processes.

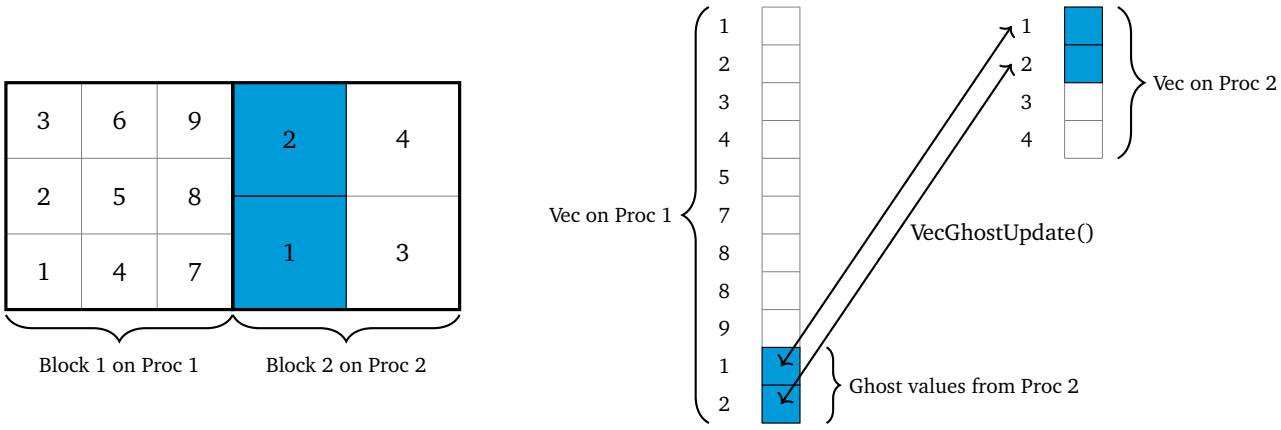


Figure 6.2: Storage and update of ghost values in the vectors related to the variables on multi-block domains. The blocks have been assigned to two different processes, Proc 1 and *Proc* 2. The control volumes of the two-dimensional problem domain are indexed with respect to the process local indexing.

The last type of data distribution refers to global reduce operations, in which a single process gathers data from all other processes and, after some modification, redistributes the data. A common scenario for global reduce operations is the calculation of the mean pressure throughout the domain which has been introduced in section 4.8.

7 Verification of the Developed CAFFA Framework

The systematical verification of program code is an essential part of the engineering process of software for scientific calculations since it ensures that the respective equations are solved correctly [**?**]. Scientific applications that solve partial differential equations can be verified using the method of manufactured solutions [**?**]. In addition, the method of manufactured solutions will be used to proof that both solvers within the developed framework, namely the segregated and the fully coupled solver, use the same discretization of the underlying partial differential equations. The current chapter presents the results of the verification process performed in the current work via the *Method of Manufactured Solutions* and begins by mentioning the main aspects of the method of manufactured solutions. The following sections present a concrete manufactured solution and the results of using it within the verification process. Finally, this chapter shows the results of a grid convergence study in order to compare the pressure weighted interpolation method from section 4.2 with the standard Rhie-Chow interpolation.

7.1 The Method of Manufactured Solutions for Navier-Stokes Equations

The method of manufactured solutions comprises a systematical, formal procedure for code verification which is based on analytic solutions to the partial differential equations to be solved [**?**]. Using these analytic solutions, the accuracy of the produced results can be assessed and, after the establishment of an acceptance criterion, used to verify the computer program. A common acceptance criterion is the *Order-of-Accuracy* criterion since, in addition to the formal order of accuracy of an implemented solver algorithm, it verifies its consistency. Careful consideration should be given to the fact that it is not possible to detect errors in the physical model by the method of manufactured solutions.

The basic idea of the method of manufactured solutions is the inversion of the solution process of partial differential equations. Instead of trying to find the solution x to a given equation $F(x) = b$ with source term b, a solution x is *manufactured* and the source term is constructed by applying F to x. This procedure has the advantage of choosing a solution that exercises all parts of the solution process and hence provides a thorough testing environment. On the downside, the integration of boundary conditions other than Dirichlet boundary conditions might be challenging [**?**]. Furthermore, the program code has to be able to handle arbitrary source terms.

The different guidelines which have to be followed in order to apply the method of manufactured solutions successfully can be found in [**?**]. According to the reference, manufactured solutions should be composed of infinitely often differentiable analytic functions whose derivatives are bounded by small constants. Furthermore, the solution domain should not be chosen to be symmetric, but as arbitrary as possible within the limitations of the code. This recommendation may, however, conflict with the application of the method of manufactured solutions to verify a code solving the Navier-Stokes equations. Section 7.2 presents the manufactured solution including the resulting source terms for the set of partial differential equations (2.7).

The verification of a solver for incompressible Navier-Stokes equations comes with additional conditions that have to be considered. If the solver is not able to handle source terms in the continuity equation and respectively the pressure-correction equation, a velocity field should be chosen that is inherently divergence-free. This property can be achieved by defining the velocity field $\mathbf{u}$ through the curl of a vector field $\mathbf{\Psi}$ as

$$\mathbf{u} = \nabla \times \mathbf{\Psi}.$$

Using the property that the divergence of the rotation of a vector field vanishes, one gets

$$\nabla \cdot \mathbf{u} = \nabla \cdot (\nabla \times \mathbf{\Psi}) = 0,$$

i.e. a locally divergence-free velocity field. This fact implies that the continuity equation is, in an integral sense, also fulfilled globally for arbitrary domains of integration. This statement is not necessarily true in the discrete sense, as integrals are to be approximated by, in the case of the present thesis, the midpoint rule of integration. In the general case for arbitrary solution domains, global mass conservation cannot be guaranteed even though the integrands are known quantities at the domain boundaries. Thus, the problem domain should be fixed before manufacturing a solution. The normal velocity field for the manufactured solution should then either vanish on the domain boundaries, which is the case of the commonly used Taylor-Green vortex [**?**], or should exhibit further symmetry that leads to cancelling non-zero mass fluxes across the domain boundaries. The studies performed during this thesis showed that convergence of the

pressure-correction equation can no longer be guaranteed if the velocity field does not obey continuity in the discrete sense at the domain boundaries.

7.2 Manufactured Solution for the Navier-Stokes Equations and the Temperature Equation

This section presents the manufactured solution for the system of partial differential equations (2.7). The manufactured solution and the computations needed to derive the respective source terms were performed by the computer algebra system Maple ® [**?**]. After the computation, the terms were directly translated into FORTRAN source code, such that the manufactured solution could be directly integrated into the solver framework. In the following paragraphs, the used solutions for the velocity vector (u_1, u_2, u_3), the pressure p and the temperature T will be formulated. The manufactured solution for the velocities was formulated as

$$
\begin{aligned}
u_1 &= 2\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_2 + 2\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_3 \\
u_2 &= 2\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_3 - 2\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_1 \\
u_3 &= -2\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_1 - 2\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_2.
\end{aligned}
$$

The manufactured solution for the pressure p was created as

$$
p = \sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right)\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right).
$$

Since the velocity field was created as a solenoidal field, no pressure or pressure-correction source had to be calculated. The manufactured solution for the temperature T reads

$$
T = \sin\left({x_1}^2\right)\cos\left({x_2}^2\right)\sin\left({x_3}^2\right).
$$

The resulting source terms for the momentum balances read

$$
\begin{aligned}
b_1 = {} & 8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_1}^2 x_2 + 8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_2}^3 + 8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_3}^2 x_2 \\
& + 8\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_1}^2 x_3 + 8\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_2}^2 x_3 + 8\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_3}^3 \\
& + 7\sin\left({x_1}^2\right)\cos\left({x_2}^2\right)\sin\left({x_3}^2\right) + 4\left(\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right)\right)^2 x_1 + 4\left(\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right)\right)^2 x_3 \\
& - 4\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right)\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_2 - 20\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_3 + 20\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_2 - 6\,x_1 \\
b_2 = {} & -8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_1}^3 + 8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_1}^2 x_3 - 8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_2}^2 x_1 \\
& - 8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_3}^2 x_1 + 8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_2}^2 x_3 + 8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_3}^3 \\
& + 13\sin\left({x_1}^2\right)\cos\left({x_2}^2\right)\sin\left({x_3}^2\right) - 4\left(\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right)\right)^2 x_2 - 4\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right)\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_1 \\
& - 4\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right)\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_3 - 20\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_1 + 20\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_3 - 2\,x_2 \\
b_3 = {} & -8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_1}^2 x_2 - 8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_2}^3 - 8\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_3}^2 x_2 \\
& - 8\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_1}^3 - 8\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_2}^2 x_1 - 8\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right){x_3}^2 x_1 \\
& - 19\sin\left({x_1}^2\right)\cos\left({x_2}^2\right)\sin\left({x_3}^2\right) + 4\left(\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right)\right)^2 x_1 + 4\left(\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right)\right)^2 x_3 \\
& - 4\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right)\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_2 + 20\cos\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_1 - 20\sin\left({x_1}^2 + {x_2}^2 + {x_3}^2\right) x_2 - 6\,x_3.
\end{aligned}
$$

Finally, the source term for the temperature equation was calculated as

$$
\begin{aligned}
b_T = &-4\sin\left({x_1}^2\right)\cos\left({x_2}^2\right)\cos\left({x_3}^2\right)\cos\left({x_1}^2+{x_2}^2+{x_3}^2\right)x_2x_3 - 4\sin\left({x_1}^2\right)\cos\left({x_2}^2\right)\cos\left({x_3}^2\right)\sin\left({x_1}^2+{x_2}^2+{x_3}^2\right)x_1x_3 \\
&+4\sin\left({x_1}^2\right)\sin\left({x_3}^2\right)\sin\left({x_2}^2\right)\cos\left({x_1}^2+{x_2}^2+{x_3}^2\right)x_1x_2 - 4\sin\left({x_1}^2\right)\sin\left({x_3}^2\right)\sin\left({x_2}^2\right)\cos\left({x_1}^2+{x_2}^2+{x_3}^2\right)x_2x_3 \\
&+4\cos\left({x_2}^2\right)\sin\left({x_3}^2\right)\cos\left({x_1}^2\right)\cos\left({x_1}^2+{x_2}^2+{x_3}^2\right)x_1x_2 + 4\cos\left({x_2}^2\right)\sin\left({x_3}^2\right)\cos\left({x_1}^2\right)\sin\left({x_1}^2+{x_2}^2+{x_3}^2\right)x_1x_3 \\
&+4{x_1}^2\sin\left({x_1}^2\right)\cos\left({x_2}^2\right)\sin\left({x_3}^2\right) + 4\sin\left({x_1}^2\right)\cos\left({x_2}^2\right){x_2}^2\sin\left({x_3}^2\right) \\
&+4{x_3}^2\sin\left({x_1}^2\right)\cos\left({x_2}^2\right)\sin\left({x_3}^2\right) - 2\sin\left({x_1}^2\right)\cos\left({x_2}^2\right)\cos\left({x_3}^2\right) \\
&+2\sin\left({x_1}^2\right)\sin\left({x_3}^2\right)\sin\left({x_2}^2\right) - 2\cos\left({x_2}^2\right)\sin\left({x_3}^2\right)\cos\left({x_1}^2\right).
\end{aligned}
$$

7.3 Measurement of Error and Calculation of Order

This section presents the results of the verification process via the formerly in section 7.2 proposed manufactured solution. To verify the solver framework using the Order-of-Accuracy test criterion, an error measure has to be chosen upon which the evaluation is based. According to [**?**] the error measure for each variable ϕ will be calculated at the end of each computation as the normalized L_2-norm of the deviation of the exact solution $\tilde{\phi}$, sometimes referred to as the normalized global error or the RMS-error

$$
\mathrm{err}(\phi) = \sqrt{\frac{\iiint_V \left(\phi - \tilde{\phi}\right)^2 \mathrm{d}V}{\iiint_V \mathrm{d}V}} \approx \sqrt{\frac{\sum_{n=1}^{N}\left(\phi_n - \tilde{\phi}_n\right)^2 V_n}{\sum_{n=1}^{N} V_n}}.
$$

Here, N denotes the number of control volumes of the discretized solution domain V. Each control volume has the volume V_n. The numerical and the exact solution are evaluated at the center of each control volume for the error calculations. In order to calculate the normalized global error of the pressure p, the analytic solution $\tilde{p}$ is modified as in section 4.8

$$
\mathrm{mean}(\tilde{p}) = \frac{\iiint_V \tilde{p}\,\mathrm{d}V}{\iiint_V \mathrm{d}V} \approx \frac{\sum_{n=1}^{N}\tilde{p}_n V_n}{\sum_{n=1}^{N} V_n}
$$

and to calculate the error as

$$
\mathrm{err}(p,N) \approx \sqrt{\frac{\sum_{n=1}^{N}\left(p_n - \tilde{p}_n - \mathrm{mean}(\tilde{p})\right)^2 V_n}{\sum_{n=1}^{N} V_n}}.
$$

Tables 7.1, 7.2 and **??** will list the calculated error terms for the velocities, the pressure and the temperature for different grid resolutions and both solver algorithms on a problem domain in the form of a cube with side length $2m$ whose center is located at $x_C = (0,0,0)$. All involved material properties are chosen to have the value 1, the gravitational acceleration is chosen to $\mathbf{g} = (7,13,-19)\,\frac{m}{s^2}$. As a reference temperature $T_0 = 0K$ is used. The grid resolutions $(n \times n \times n)$ are chosen in a way that n, the number of control volumes in each coordinate direction, will be a power of 2. The errors are evaluated after the maximal relative residual of all linear systems in one outer iteration fell below the threshold 1×10^{-14}. The concrete implementation of the tolerance criterion to accept a converged solution is found in section 6.3.2. Due to convergence problems of the linear equation solver, no results were calculated with the coupled solver for the case of $256 \times 256 \times 256$ unknowns.

A comparison of the presented results leads to the first conclusion that the calculated errors from the segregated solver algorithm and the coupled solver algorithm coincide. This result shows that both implementations use the same discretization. According to [**?**] the next step comprises the calculation of the formal order of accuracy $\hat{p}_\phi$ for each variable for which the program solves. This calculation can be carried out by evaluating the quotient of two different grid resolutions $n1$ and $n2$ as

$$
\hat{p}_\phi = \frac{\log\left(\frac{\mathrm{err}(\phi,n1)}{\mathrm{err}(\phi,n2)}\right)}{\log\left(\frac{n1}{n2}\right)}.
$$

The discretization techniques used in the present thesis yield an asymptotic discretization error of 2 [**?**]. A comparison of the results in Table 7.1, Table 7.2, and Table **??** shows the asymptotic behavior of the calculated order which leads to the conclusion that the discretization has been implemented successfully.

Table 7.1: Comparison of the errors of the velocity and the resulting order of accuracy, calculated by the segregated and the coupled solver for different grid resolutions

Resolution	Error of **u** from segregated solver	Error of **u** from coupled solver	Observed order $\hat{p}$
8x8x8	3.170450230115786E-002	3.170450290288525E-002	
	3.037011030746674E-002	3.037011030619578 3E-002	
	3.163609515028456E-002	3.163609514130644 2E-002	
16x16x16	7.914356673006179E-003	7.914356673006535 0E-003	2.0021
	7.534314612440339E-003	7.534314612440905 6E-003	2.0111
	7.811316741535871E-003	7.811316741536102 1E-003	2.0179
32x32x32	1.933302886747634E-003	1.933325081409706 3E-003	2.0334
	1.867599197138540E-003	1.867651252317599 0E-003	2.0123
	1.894813910088982E-003	1.894822320599556 2E-003	2.0435
64x64x64	4.746894199493598E-004	4.746894200152712 4E-004	2.0260
	4.629259933395594E-004	4.629259934877650 2E-004	2.0123
	4.630008203550293E-004	4.630008203802782 2E-004	2.0330
128x128x128	1.172266972290949E-004	1.172266974040543 E-004	2.0177
	1.150024026498247E-004	1.150024030604766 E-004	2.0091
	1.140471182846354E-004	1.140471183418716 E-004	2.0214
256x256x256	2.876932445431025E-005		2.0267
	2.771472789328757E-005	d.n.c.	2.0529
	2.826736500332136E-005		2.0124

Table 7.2: Comparison of the errors of the pressure and the resulting order of accuracy, calculated by the segregated and the coupled solver for different grid resolutions

Resolution	Error of p from segregated solver	Error of p from coupled solver	Observed order $\hat{p}$
8x8x8	0.183035893121686	0.183035894664680	
16x16x16	8.967727683983406E-002	8.967727683982576E-002	1.0293
32x32x32	3.796485688816108E-002	3.796481581400934E-002	1.2401
64x64x64	1.438780100622541E-002	1.438780100615654E-002	1.3998
128x128x128	5.160705648852031E-003	5.160705649392788E-003	1.4792
256x256x256	1.809811107444374E-003	d.n.c.	1.5117

7.4 Influence of the Under-Relaxation Factor for the Velocities

In section 4.2, the main purpose of introducing the pressure weighted interpolation method, was to assure comparability of the results generated by the segregated solver algorithm and the coupled solver algorithm. The errors presented in section 7.3 show good agreement between both solvers. However, the segregated solver was using the pressure weighted interpolation method. This section presents the results that are attained, if instead of the pressure weighted interpolation method the standard Rhie-Chow interpolation technique is applied. Figure **??** shows the normalized L_2-norm of the error of the results for the velocity u_1 for different under-relaxation factors. The L_2-norms have been normalized by the L_2-norm of the respective calculation that uses the pressure weighted interpolation method.

Table 7.3: Comparison of the errors of the temperature and the resulting order of accuracy, calculated by the segregated and the coupled solver for different grid resolutions

Resolution	Error of T from segregated solver	Error of T from coupled solver	Observed order $\hat{p}$
8x8x8	4.067532377541200E-003	4.067532377537829E-003	
16x16x16	1.132477318394773E-003	1.132477318394782E-003	1.8447
32x32x32	2.931009801582052E-004	2.931009624340290E-004	1.9500
64x64x64	7.410021974469997E-005	7.410021974334421E-005	1.9838
128x128x128	1.858846421984987E-005	1.858846421622182E-005	1.9951
256x256x256	4.651911366447223E-006	d.n.c.	1.9985

Figure **??** shows that the higher the under-relaxation factor, the smaller the deviation of the results obtained with the standard Rhie-Chow interpolation technique from the results obtained with the pressure weighted interpolation method. These results are in agreement with the fact that, according to equation (4.5) the pressure weighted interpolation and the Rhie-Chow interpolation yield the same results, when an under-relaxation factor $\alpha_{\mathbf{u}} = 1$ is used. Furthermore, Figure **??** shows that the deviation of both methods decreases with increasing mesh resolution which is the expected behavior [**?**].

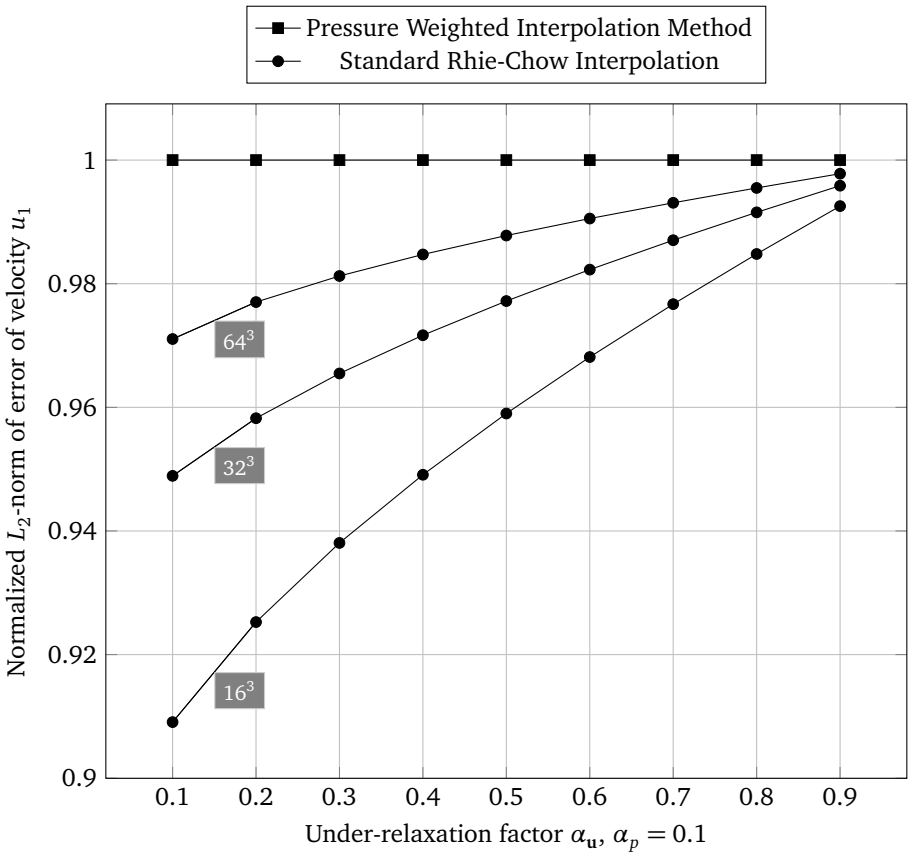


Figure 7.1: Comparison of the calculated relative error of the u_1 velocity for different under-relaxation factors $\alpha_{\mathbf{u}}$ on grids with $16 \times 16 \times 16$, $32 \times 32 \times 32$ and $64 \times 64 \times 64$ cells

8 Comparison of Solver Concepts

This chapter presents the performance analyses of both developed solvers within the CAFFA framework. In the first part of this chapter, the used hardware and software is described. Furthermore, necessary performance measures are introduced. After showing the outcomes of a preliminary benchmark to measure the sustainable bandwidth, the results of the Speed-Up and the weak scalability study are presented. The second part of this chapter analyzes the performance of the algorithms on a test case with a complex geometry and a test case involving natural convection. Furthermore, the results of a study on the effect of non-orthogonal corrections on the convergence behavior of the coupled solution algorithm are presented. The last section of this chapter proposes a method for load balancing by automatic matrix partitioning.

8.1 Parallel Performance

In many cases, solver applications are used to solve complex problems regarding memory requirements and are, at the same time, expected make calculation results available within a short time. In both scenarios, code that designed to run in parallel can address the mentioned challenges. Code that runs in parallel can allocate more memory resources which makes the calculation of complex problems feasible. If the code is scalable, the program execution can be shortened by involving more processes to solve a problem of constant size on the compatible hardware. The CAFFA solver framework, which has been extended in the course of the present thesis, has been parallelized using the PETSc library. The purpose of this section is to assess the parallel performance of this solver framework on a high-performance computer.

After introducing the used hardware and software, this section presents central measures of parallel performance. Then, the outcome of a preliminary low-level benchmark is presented. This benchmark was carried out in order to establish upper performance-bounds on the parallel efficiency and the scalability of the developed solver framework. The last sections present the results of the parallel efficiency evaluation of the solver framework.

8.1.1 Employed Hardware and Software – The Lichtenberg-High-Performance Computer

All performance analyses which are presented in this thesis build on tests conducted on the Lichtenberg-High-Performance Computer, also known as *HHLR* (*Hessischer Hochleistungsrechner*). The cluster consists of different sections according to the used hardware. Throughout the thesis, tests were performed using the first (MPI1) and the second MPI (MPI2) section of the cluster. The first section (MPI1) consists of 705 nodes of which each runs two Intel®Xeon®E5-2670 processors and offers 32GB of memory. The second section consists of 356 nodes of which each runs two Intel Xeon E5-2680 v3 processors and offers 64GB of memory. As interconnect for both sections, FDR-14 InfiniBand is used.

All tests programs were compiled using the Intel compiler suite version 15.0.0 and the compiler options

−O3 −xHost.

As MPI implementation, Open MPI version 1.8.2 was chosen. Furthermore, the PETSc version 3.5.3 was configured using the options

−−with−blas−lapack−dir=/shared/apps/intel/2015/composer_xe_2015/mkl/lib/intel64/ \
−−with−mpi−dir=/shared/apps/openmpi/1.8.2_intel \
COPTFLAGS="−O3 −xHost" \
FOPTFLAGS="−O3 −xHost" \
CXXOPTFLAGS="−O3 −xHost" \
−−with−debugging=0 \
−−download−hypre \
−−download−ml

To maximize the efficiency of PETSc, a math kernel library should be used that has been optimized for the underlying hardware architecture. In the case of the present thesis the Intel *MKL* (*Math Kernel Library*) was chosen. The Open MPI library was compiled using Intel compilers as well.

8.1.2 Measures of Performance

This section establishes the needed set of measures to evaluate the performance of a solver program which will be used in the following sections. The first measure is the plain measure of the computer's runtime T_P for solving a given problem, where $P \in \mathbb{N}$ denotes the number of involved processes. This so-called *wall-clock* time can be measured directly by calling subroutines of the underlying operating system and corresponds to the human perception of the time which has passed. This time does not correspond to the often mentioned *CPU* time. In fact, CPU time is only one contributor to wall-clock time. On the one hand, wall-clock time contains the time needed for communication and I/O, hence considers idle states of the processor. On the other hand, CPU time only considers the time in which the processor is actively working. This fact makes wall-clock time not only a more complete time measure, but also a more accurate one when dealing with parallel processors. The idle times of the processor due to communication are taken into account by the wall-clock time, whereas they are ignored by CPU time.

While wall-clock time is an absolute measure that can be used to compare different solver programs, further relative measures are needed to evaluate the efficiency of one program regarding the parallelization. The main purpose of these measures is to rate the different causes of degrading efficiency due to parallelization of the algorithm or the parallel execution of the algorithm. A simple model [**?**, **?**] considers three contributions which form the total efficiency

$$E_P^{tot} = E_P^{num} \cdot E_P^{par} \cdot E_P^{load}. \tag{8.1}$$

The *numerical efficiency*

$$E_P^{num} := \frac{\text{FLOPS}(1)}{P \cdot \text{FLOPS}(P)}$$

considers the degradation of the efficiency of the underlying algorithm due to the parallelization. Many efficient algorithms owe their efficiency to recursions inside the algorithm. In the process of resolving these recursions, the efficiency of the algorithm degrades. It follows that this efficiency is entirely independent of the underlying hardware.

The *parallel efficiency*

$$E_P^{par} := \frac{\text{TIME(parallel algorithm on one processor)}}{P \cdot \text{TIME(parallel algorithm on } P \text{ processors)}}$$

describes the effect of the need for inter-process communication, if the solution process involves more than one process. This form of efficiency does explicitly exclude any algorithm related degradation since the time measured corresponds to the same algorithms. It follows that the parallel efficiency only depends on the implementation of the communication and the latencies associated with hardware.

The *load balancing efficiency*

$$E_P^{load} := \frac{\text{TIME(calculation on complete domain)}}{P \cdot \text{TIME(calculation on biggest subdomain)}}$$

is formed by the quotient of the wall-clock times needed for the complete problem domain and partial solves on subdomains. This measure does neither depend on hardware nor the used implementation. Instead, it directly relates to the size and partition of the grid.

It is not possible to calculate all three efficiencies at the same time using only plain wall-clock time measurements of a given application. Different solver configurations have to be used to calculate them separately. Since the focus of the investigation of the present thesis does not lie on load balancing, for all further tests that do not directly address load balancing $E_P^{load} = 100\%$ is assumed. This assumption does not present a considerable drawback since an ideal load balancing is easily obtainable nowadays by the use of grid partitioning algorithms [**?**]. Section **??** presents a simple method to achieve load balancing for the linear solvers. Using identical algorithms for different numbers of involved processes, implicitly achieves $E_P^{num} = 100\%$. In this case, the parallel efficiency of an application can be measured through the quotient of the needed wall-clock time. In order to measure the numerical efficiency of an algorithm, the respective hardware counters have to be evaluated. This evaluation can be made using the built-in log file functionality of PETSc.

Another common performance measure is the *Speed-Up*

$$S_P = \frac{T_1}{T_P} = P \cdot E_P^{tot}.$$

Speed-Up and parallel efficiency characterize the parallel scalability of an application and determine the regimes of efficient use of hardware resources.

8.1.3 Determining Upper Performance-Bounds – The STREAM Benchmark

Scientific applications which solve partial differential equations rely on sparse matrix computations, which usually exhibit the sustainable memory bandwidth as bottleneck with respect to the runtime performance of the program [**?**]. The purpose of this section is to establish a frame in terms of an upper performance-bound in which the efficiency of the developed solver framework can be evaluated critically. As a standard measure for the maximum sustainable bandwidth, low-level benchmarks can be used which focus on evaluating specific properties of the deployed hardware architecture. In this case, the Mc Calpin STREAM benchmark suite [**?**, **?**] provides apt tests which are designed to work with data sets that exceed the cache size of the involved processor architecture. Consequently the benchmark forces the processors to stream the needed data directly from memory instead of reusing the data residing in their caches. These types of tests can be used to calculate an upper bound on the memory bandwidth for the CAFFA framework.

In particular this subsection presents the calculated sustainable memory bandwidth as determined by the TRIAD kernel of the STREAM benchmark suite. The kernel loop of the TRIAD kernel consists of the operation

$$a[i] = b[i] + scalar * c[i],$$

where the sizes of the vectors a, b, and c have to be chosen according to the deployed hardware.

In terms of parallel scalability, the STREAM benchmark can also be used to determine an upper performance-bound. According to [**?**], the parallel performance of memory bandwidth limited codes correlates with the parallel performance of the STREAM benchmark, i.e. a scalable increase in memory bandwidth is necessary for scalable application performance. The intermediate results of the benchmark can then be used to test different configurations that bind hardware resources to the involved processes. Figure **??** and Figure **??** show the test results of the STREAM benchmark on the two different MPI sections of the HHRL. For the results to be reproducible, it is central to run these tests exclusively on each node, i.e. that no other test is running on the same node.

As can be seen from Figure **??** and Figure **??**, the scaling of the sustainable bandwidth behaves rather erratic. As a consequence, for process counts up to eight for the MPI1 section and process counts up to six for the MPI2 section, no reliable results can be obtained from the STREAM benchmark. It is assumed that this kind of behavior is due to the automatic turbo-boost the deployed processors apply, which cannot be controlled other than by turning it off on the nodes directly from the BIOS. For the following performance analyses with respect to parallel performance, it is desirable to minimize this kind of unpredictable side effects. Since all performance measures are relative, with the exception of the plain measurement of wall-clock time, the reference value for the following relative performance measurements will be taken from the program execution for the maximum number of processes that can be bound to one deployed socket without any overlapping.

The STREAM benchmark used to test the present architecture has been provided by the PETSc framework. Special effort has been made to adapt this benchmark to handle the described problem. This was done by always using all available cores per socket, but running the benchmark only with n processes, while the other $(16-n)$ or $(24-n)$ processes are running calculations which are only using data from the L1-cache of their corresponding core. The intention of this approach was to regulate the turbo-boost effects by keeping a constant load on the processors for different numbers of STREAM benchmark processes. However, this approach did not generate results different from Figure **??** and Figure **??**.

8.1.4 Speed-Up Measurement and Effect of Coupling Algorithm for Analytic Test Case

This section presents the results from the Speed-Up measurements conducted on the supercomputer HHLR which has been presented in section **??**. The tests for this section distribute a problem of constant size to an increasing number of processes. The measurements of the decrease of runtime will show if the application performance improves by involving more processes in the solution process. This property of applications if often termed *strong* scalability [**?**]. Furthermore, the effect of the coupling algorithm on the running time of the corresponding solver program will be shown.

All Speed-Up tests are conducted on the MPI1 section. The performance analyses will show the absolute time needed to solve for the flow of the analytic solution presented in section 7.2. In order to evaluate the parallel performance, the Speed-Up will be calculated based on the presented results. Since the performance of segregated algorithms depends on the amount of under-relaxation applied to the variables in each outer iteration, a preliminary test is conducted to support choosing the optimal amount of under-relaxation. As [**?**] and [**?**] point out, it is recommended to choose the amount of under-relaxation for the pressure depending on the amount of under-relaxation for the velocity as

$$\alpha_p = 1 - \alpha_{\mathbf{u}}.$$

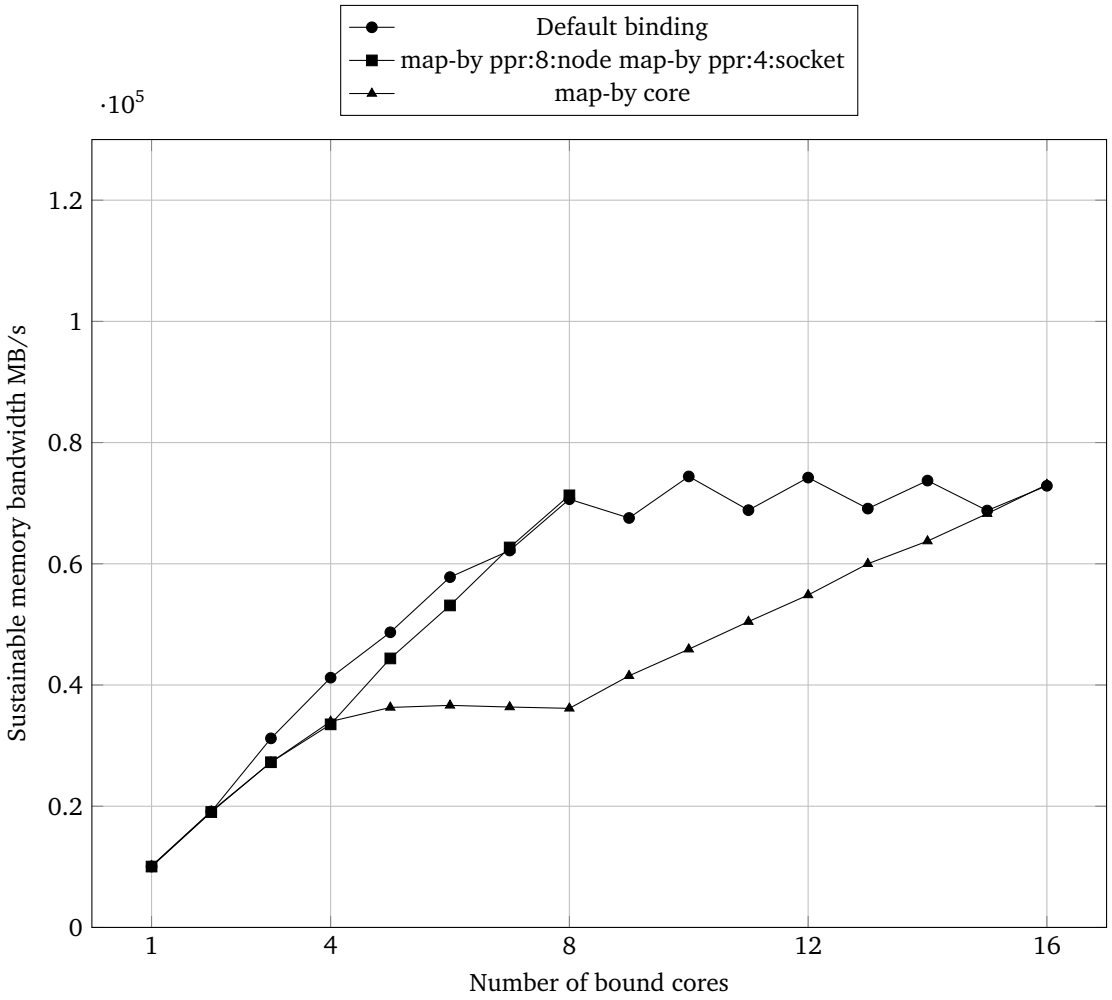


Figure 8.1: Sustainable memory bandwidth as determined by the STREAM benchmark (TRIAD) for different Open MPI process binding options on one node of the MPI1 section of the HHLR

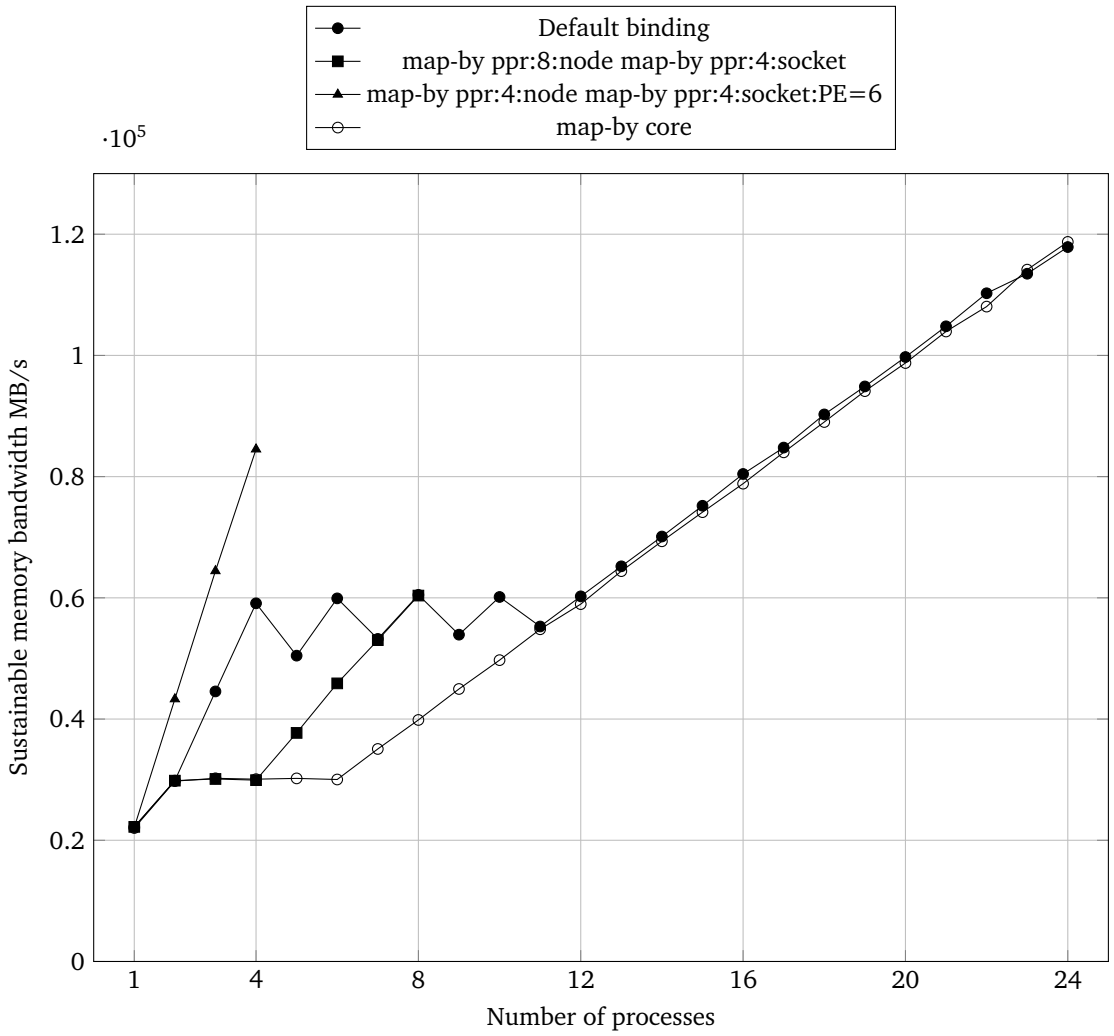


Figure 8.2: Sustainable memory bandwidth as determined by the STREAM benchmark (TRIAD) for different Open MPI process binding options on one node of the MPI2 section of the HHLR

Figure **??** shows the results of the preliminary runtime measurements for different choices of the under-relaxation factor $\alpha_{\mathbf{u}}$. The problem which was solved in this test corresponds to the manufactured solution for the velocities and the pressure presented in section 7.2. Figure **??** shows that the optimal under-relaxation factor can be found in a neighborhood of 0.95. Based on Figure **??** the under-relaxation was chosen to $\alpha_{\mathbf{u}} = 0.9$ for the following performance analysis.

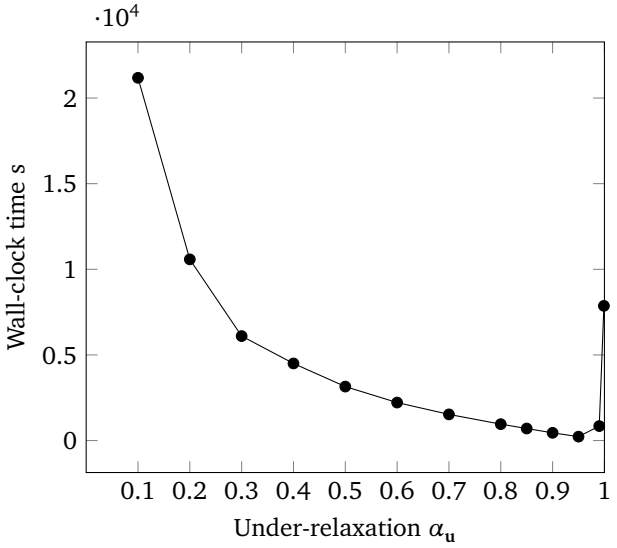


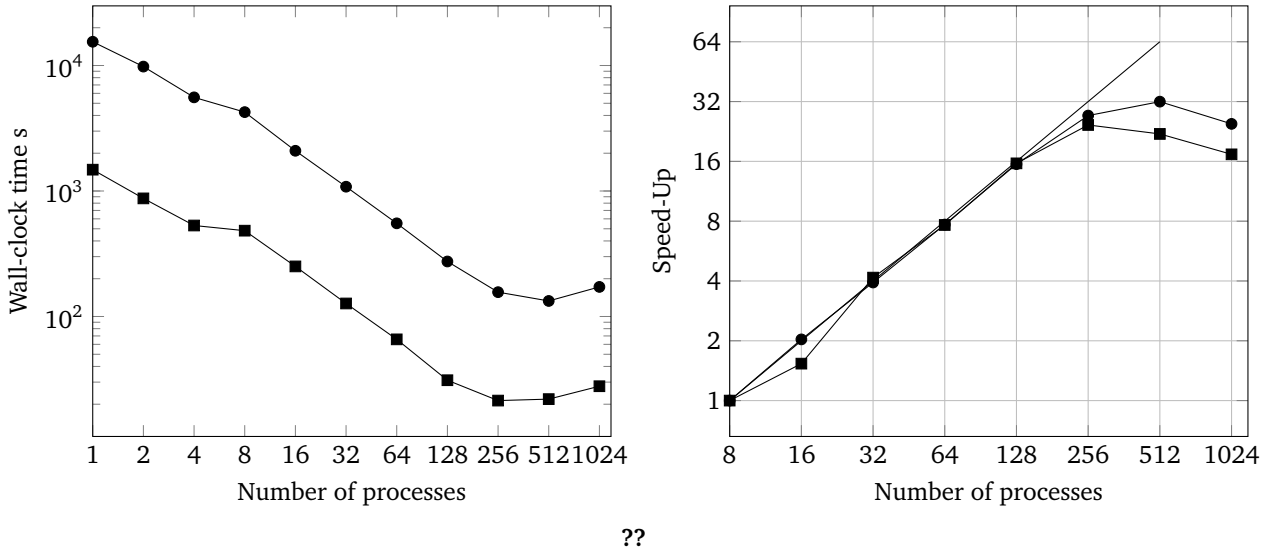
Figure 8.3: Wall-clock time comparison for the SIMPLE-algorithm, solving for the analytic solution presented in section 7.2 on a structured grid with $64 \times 64 \times 64$ cells using different under-relaxation factors up to $\alpha_{\mathbf{u}} = 0.999$

The performance of all solvers significantly depends on the performance of the linear equation solvers and the corresponding preconditioners used in the solution process. For all conducted tests, the linear system of the discretized momentum balances will be solved with a GMRES (*Generalized Minimal Residual*) [**?**] Krylov subspace solver and an incomplete LU-factorization as preconditioner. The pressure-correction equation is solved by a CG (*Conjugated Gradient*) solver [**?**] with the GAMG (*Geometric Algebraic Multi Grid*) preconditioner from PETSc. The coupled systems are solved by a GMRES Krylov subspace solver with GAMG preconditioning. The GAMG preconditioner was chosen so that a high numerical efficiency is obtained, constituting a basis for further parallel performance measurements based on plain wall-clock time measurements. Table **??** lists all further solver parameters.

Table 8.1: Characteristic problem properties used in the performance measurements, solving for the analytic solution presented in section 7.2

Property	Value	Unit
Density	1E-0	kg/m^3
Viscosity	1E-0	Ns/m^2
Under-relaxation u	0.9	
Under-relaxation p	0.1	
Relative tolerance	1E-8	

From the comparison of both algorithms in Figure **??** it is evident that both algorithms are scalable, and that the coupled solution algorithm accelerates the solution process almost up to one order of magnitude. For higher amounts of under-relaxation for the velocities the difference in performance is expected to increase as Figure **??** suggests. The presented results for the coupled solution algorithm were obtained using black-box solvers from the PETSc framework. It is assumed that the performance of the coupled solution algorithm with respect to the needed wall-clock time would benefit from multigrid solvers especially designed for the coupled system of linear equations. Examples for such special solvers can be found in [**?**, **?**, **?**]. *Physics-based* solvers represent another approach. These solvers take advantage of the matrix structure as shown in figure 5.3. References [**?**, **?**] describe how PETSc can be used to construct physics based preconditioners in the context of multiphysics simulations.



??

Figure 8.4: Wall-clock time and Speed-Up comparison for the segregated and the fully coupled solution algorithm, solving for the analytic solution presented in section 7.2 on a structured grid with $128 \times 128 \times 128$ cells

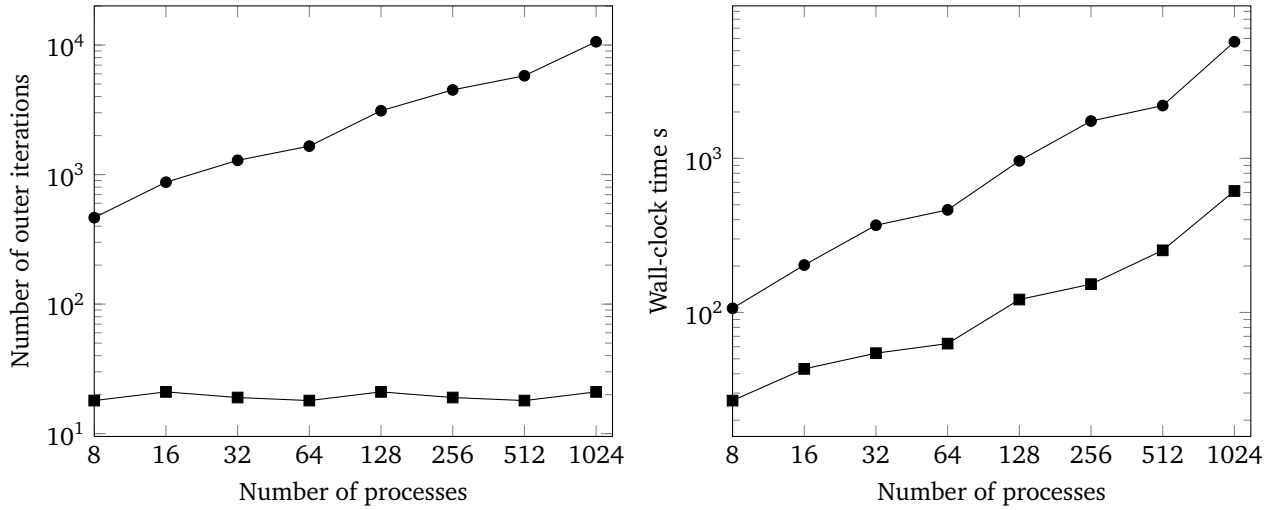
8.1.5 Weak Scaling Comparison of Coupled and Segregated Solution Algorithm

References [**?**] and [**?**] report that the coupled solution algorithm also scales well with the size of the problem domain with respect to the involved number of unknowns. Generally, this is not true for the SIMPLE-algorithm as representative of the set of segregated solution algorithms. An algorithm that scales well with the problem size is apt to be tested on a benchmark that analyzes the so-called *weak* scaling of the implementation [**?**]. This benchmark will investigate how the running time and the number of needed inner and outer iterations behave when the computational load per process is held constant while the number of involved processes is continuously increased. As test problem, the flow with the analytic solution presented in 7.2 was chosen. *Weak* scaling benchmarks show, if it is possible to solve a more complex problem in the same amount of time by involving more processes running the solver application. Each process will get assigned a block with $32x32x32$ unknowns. Thus, the number of involved unknowns N in each test run can be calculated as $N = 32 \times 32 \times 32 \times n$, where n denotes the number of involved processes. Figure **??** compares the needed number of outer iterations and the corresponding wall-clock time for an implementation of the SIMPLE-algorithm and the fully coupled solution algorithm.

The comparison in Figure **??** shows that the fully coupled algorithm scales ideally with respect to the number of outer iterations, which remains almost constant throughout the conducted study. The number of iterations at most differed by two outer iterations. These differences are suspected to be an effect of the loose tolerance criterion of residual reduction in each outer iteration. As an effect of accumulating rounding errors, some test cases did not accomplish the total tolerance criterion and thus needed to perform another additional outer iteration. From the wall-clock time measurements presented in Figure **??** it is evident that, despite the approximately constant number of outer iterations, the fully coupled solution algorithm did not scale ideally. Figure **??** shows that weak scaling is prevented because the linear solver does not scale ideally. This fact shows that weak scaling cannot be obtained using black-box linear solvers. Further research regarding the development of special linear solvers for the systems resulting from a fully coupled discretization of the Navier-Stokes equations is needed.

8.2 Realistic Testing Scenario – Complex Geometry

Fluid flow inside closed applications is a common situation in mechanical engineering. The flow through a channel with rectangular cross section represents a simple test case, which is a part of complex applications. This section compares the single process performance of the segregated and the fully coupled solution algorithm for a flow problem within a complex geometry. The geometry of the domain is based on a channel flow problem with square cross section. Inside the channel reside two obstacles with a square cross section of which one has been twisted against the other by 45°. Figure **??** shows a sketch of the problem domain.



??

Figure 8.5: Comparison of the number of outer iterations and the corresponding wall-clock time, needed to achieve a relative reduction of 1E-8 of the initial residual, for different methods to resolve pressure-velocity coupling

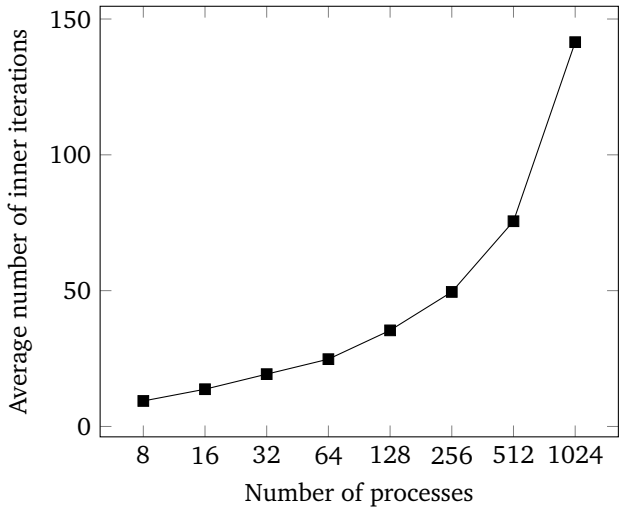


Figure 8.6: Average number of inner iterations for different process counts solving for the analytic solution presented in section 7.2 with the fully coupled solution algorithm

This test case utilizes all of the previously introduced boundary conditions for flow problems, including the use of non-matching block boundaries. For the velocities at the inflow boundary, the parabolic distribution

$$\mathbf{u}(x_1, x_2, x_3) = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} \dfrac{16 * 0.45 * x_2 * x_3 * (0.41 - x_2) * (0.41 - x_3)}{0.41^4} \\ 0 \\ 0 \end{bmatrix}$$

was chosen. At the outlet, a Dirichlet pressure boundary condition was used. All other boundaries used a no-slip solid non-moving wall boundary condition. All problem parameters were chosen to assure that the flow problem resides in the regime of a non-turbulent stationary flow, for which the presented solver framework has been developed. Table **??** lists the remaining material and geometrical characteristics of the test case. A study similar to the one presented in section **??** was conducted to optimize the used amount of under-relaxation.

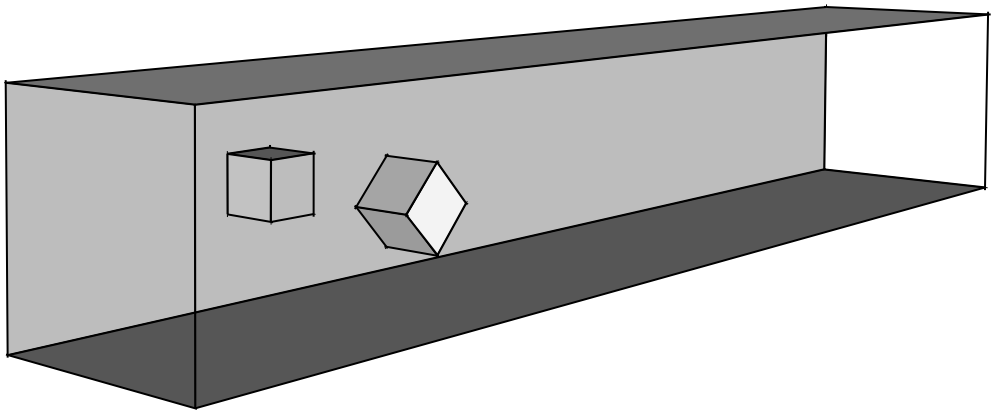


Figure 8.7: Sketch of the channel flow problem domain

Table 8.2: Characteristic problem properties used in the channel flow test case

Property	Value	Unit
Density	1E-0	kg/m^3
Viscosity	1E-3	Ns/m^2
Height	0.41	m
Length	2.5	m
Side length cube	0.1	m
Under-relaxation u	0.9	
Under-relaxation p	0.1	
Relative tolerance	1E-8	

Using the data from Table **??** and the maximal inflow velocity of $0.45m/s$ the Reynolds number Re can be calculated as

$$Re = \frac{\rho \,\mathrm{mean}(u_1) l}{\mu} = 20,$$

where $\mathrm{mean}(u_1) = 0.45 * \frac{4}{9} m/s = 0.2m/s$. This shows that the flow resides in a laminar regime.

The present test case shows one advantage of the treatment of block boundaries, which has been introduced in section 4.7.3. Since no assumptions on the geometry of a neighboring block are necessary, the mesh for each block can be constructed independently, which increases the flexibility of the meshing of geometries. Furthermore, because of the fully implicit handling of block boundaries, the number of used blocks does not negatively affect the convergence of the deployed linear solvers. Figure **??** shows the mesh on the left and right bounding walls. This mesh leads to non-trivial transitions between the blocks. Figure **??** shows the domain decomposition into structured grid blocks around the two obstacles within the problem domain and emphasizes the need for accurate handling of non-matching block boundaries.

The solution of the linear systems resulting from the discretization of the problem takes up more time during the execution of the coupled solution algorithm than during the execution of the segregated algorithm. Furthermore, the segregated solution algorithm for small numbers of unknowns does not need many outer iterations to converge. This fact makes the segregated solver the faster one for problems involving small numbers of unknowns. However, as the number of unknowns increases, the number of needed outer iterations increases due to the under-relaxation. This behavior shows that segregated algorithms do not scale with increasing problem size.

In contrast to the segregated algorithm, the fully coupled solution algorithm achieves an approximately constant amount of needed outer iterations, independent of the number of involved unknowns. The tests regarding the weak scalability of the coupled solution algorithm presented in section **??** emphasize this property. Table **??** compares the measured wall-clock time for different numbers of unknowns. Figure **??** shows the mesh generated for the first number of unknowns. The two other numbers of unknowns result from bisecting the mesh in each direction. Every bisection of the grid scales the number of unknowns by a factor of approximately eight. The tests were conducted on the formerly presented HHLR cluster, using the MPI2 section.

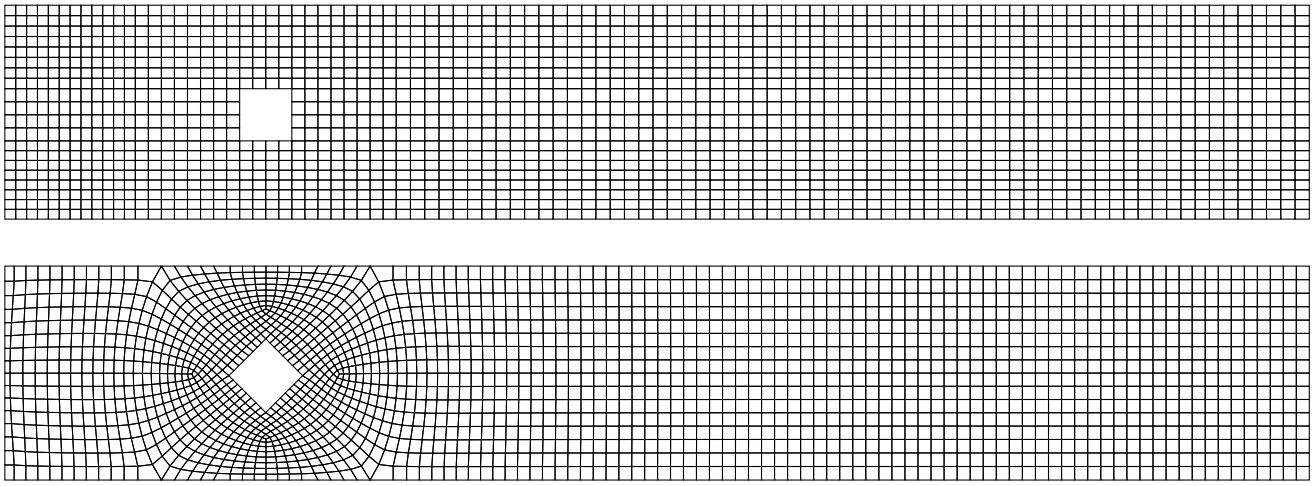


Figure 8.8: West and east boundary of the numerical grid for the channel flow problem

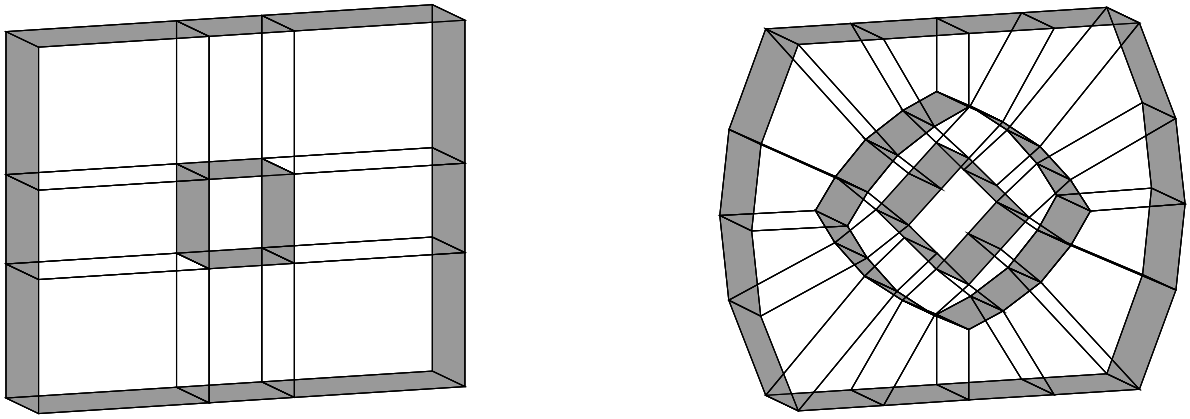


Figure 8.9: Blocking for the two different obstacles within the problem domain of the channel flow

Table 8.3: Performance analysis results of the channel flow problem for different numbers of unknowns comparing the segregated (SEG) to the fully coupled (CPLD) solution algorithm using one process on the MPI2 section of the HHLR supercomputer.

No. of unknowns	SEG - time s	CPLD - time s	SEG - its	CPLD - its
75768	0.2226E+02	0.2674E+02	151	67
408040	0.4053E+03	0.1499E+03	355	42
2611080	1.1352E+05	0.3105E+04	1592	39

The timing results show that already after the first grid refinement step the fully coupled solution algorithm performs better with respect to the needed wall-clock time for computation. This effect is clearly visible for higher mesh resolutions.

8.3 Classical Benchmarking Case – Temperature-Driven Cavity Flow

This section deals with the evaluation and comparison of the velocity-to-temperature coupling, through the Boussinesq approximation, and the temperature-to-velocity/pressure coupling, through the Newton-Raphson linearization of the convective term of the temperature equation. For this, the standard temperature-driven cavity flow [**?**, **?**] is adapted for three-dimensional domains. The material and geometrical parameters are chosen in a way that a non-turbulent and stationary flow exists. Figure **??** shows an example of a temperature and velocity field obtained with the developed solver framework, solving for the temperature-driven cavity flow problem.

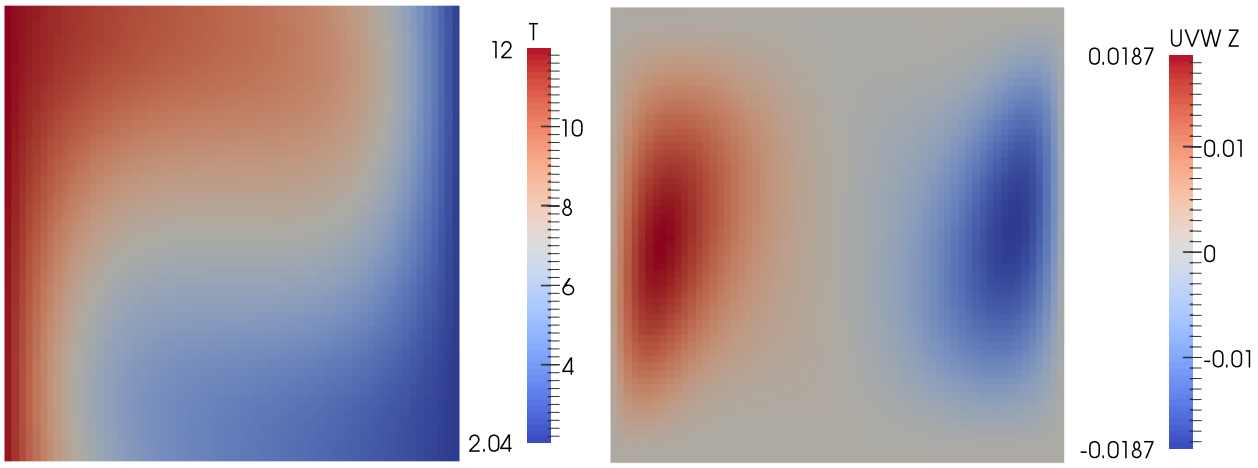


Figure 8.10: Temperature field and velocity field in the coordinate direction of the gravitational force of the temperature-driven cavity flow problem for a cross section in the middle of the cubical-shaped problem domain as $Ra = 10^4$

Essential for this benchmarking case is the nature of the flow. The fluid motion is a consequence of the effect of volume forces caused by temperature differences in the solution domain. Hence, the mathematical problem exhibits a strong coupling between the involved variables velocity, pressure and temperature. This relation is represented by the Rayleigh and Prandtl number of the problem. It is assumed that the fully implicit treatment of the temperature coupling will yield further benefits with respect to wall-clock time for this kind of flow, compared with solution approaches that solve for the temperature separately. Table **??** lists the geometrical and solver parameters used for the performance analysis of this section.

Using the data from Table **??** and given a Prandtl number for the flow problem $Pr = 0.71$, the Rayleigh number Ra can be calculated. The Rayleigh number is a dimensionless quantity used to characterize buoyancy-driven flows. From the definition of the Prandtl number, the thermal diffusivity κ can be calculated as

$$\kappa = \frac{\mu}{\rho\, Pr} = 2.1304 * 10^{-5}.$$

Table 8.4: Characteristic problem properties used in the temperature-driven cavity flow test case

Property	Value	Unit
Density	1.19	kg/m^3
Viscosity	1.8E-5	Ns/m^2
Height	0.021277	m
Temperature difference	10	K
Coefficient of thermal expansion	0.00341	$1/K$
Under-relaxation u	0.8	
Under-relaxation p	0.2	
Under-relaxation T	1.0	
Relative tolerance	1E-8	

It follows that the Rayleigh number can be determined to

$$Ra = \frac{\rho g \beta \Delta T h^3}{\mu \alpha} \approx 10^4,$$

which implies, according to [**?**], that the flow is still stationary and non-turbulent.

The tests were conducted on the formerly presented HHLR cluster, using the MPI2 section. Table **??** compares the wall-clock times and numbers of needed outer iterations for different solver and coupling approaches. The presented results are in good agreement with [**?**]. This reference shows a monotonic decrease of the number of iterations with increased implicit coupling for a different test case at a Rayleigh number similar to the one used in the present thesis. The implicit pressure-velocity coupling is responsible for the biggest decrease in the number of nonlinear iterations. Different to the results presented in section **??**, this does not yield significant performance benefits with respect to wall-clock time. In order to achieve the benefits of a fully coupled solution algorithm, the coupling has to be extended to involve semi-implicit temperature-to-velocity/pressure and implicit velocity-to-temperature coupling as well. In this context, the abbreviations *SEG* for the segregated solution algorithm and *CPLD* for the fully coupled solution algorithm without implicit temperature coupling are used. For the fully coupled solver configurations, *TCPLD* includes only implicit velocity-to-temperature coupling, whereas *NRCPLD* additionally includes implicit temperature-to-velocity/pressure coupling via the Newton-Raphson linearization.

Table **??** demonstrates that the sole use of velocity-to-temperature coupling does not result in benefits compared to the coupled solution process for velocities and pressure combined with a decoupled solve for the temperature equation. This characteristic is accredited to the treatment of the nonlinearity of the temperature equation in the TCPLD solver configuration. Even though the momentum balances implicitly use the temperature from the next iteration, the temperature equation does not use the velocities of the next iteration. Instead, the convective fluxes are calculated with the velocities from the previous iteration. Even though, the implicit velocity-to-temperature coupling reduces the number of needed nonlinear iterations Table **??** shows an increase in the required wall-clock time to finish the computations. This increase is attributed to the augmented costs during the application of the linear solver algorithm as a result of the additional degree of freedom that the linear system embraces. The necessary number of outer iterations is higher than in the NRCPLD configuration since the convective fluxes in the temperature equation are not coupled to the momentum balances, a fact that degrades the convergence of the nonlinear iteration process.

The conducted study shows that in order to profit from the benefits of implicit coupling with to the needed number of nonlinear iterations the key lies in the semi-implicit temperature-to-velocity/pressure coupling. However, the implicit consideration of the corresponding terms furthermore increases the amount of memory needed for computation and hence does not come without deficiencies.

8.4 Effect of Different Non-Orthogonal Correctors on Solver Convergence

Section 3.3 introduced different ways to address the degradation of the grid quality due to non-orthogonality of the grid. This section compares the three different correctors, the orthogonal correction, the minimum correction, and the over-relaxed approach, which are used in the discretization process of the gradients at the cell boundary faces. For this purpose the grid generator program of section 6.2 was extended to generate skewed grids. A random number generator was used to move each inner grid point within a specified neighborhood of the original location and consequently skew

Table 8.5: Performance analysis results for the temperature-driven cavity flow problem comparing the SIMPLE-algorithm with segregated temperature solve (SEG), the fully coupled solution algorithm with segregated temperature solve (CPLD), the fully coupled solution algorithm with an implicit Boussinesq approximation (TCPLD) and the fully coupled solution algorithm using an implicit Boussinesq approximation and a semi-implicit Newton-Raphson linearization of the convective part of the temperature equation (NRCPLD).

Resolution	Solver configuration	Time s	No. Nonlinear its.
32x32x32	SEG	0.3719E+02	203
	CPLD	0.6861E+02	62
	TCPLD	0.1012E+03	31
	NRCPLD	0.2153E+02	22
64x64x64	SEG	0.1997E+04	804
	CPLD	0.7687E+03	63
	TCPLD	0.1278E+04	59
	NRCPLD	0.4240E+03	17
128x128x128	SEG	0.5197E+05	3060
	CPLD	0.1860E+05	74
	TCPLD	0.1950E+05	50
	NRCPLD	0.6155E+04	18

the grid. Figure **??** illustrates the implemented concept for a two-dimensional 8×8 grid. The orthogonality of the grid is lost, and the need for non-orthogonal correctors becomes visible. Even in the case of orthogonal grids, a non-orthogonal correction may become necessary to treat fluxes across block boundaries if neighboring blocks have been locally refined. To maintain the grid's integrity after the movement of the grid vertices, the neighborhoods are limited by half the distance Δx of two neighboring grid points, as indicated for one particular vertex in Figure **??**.

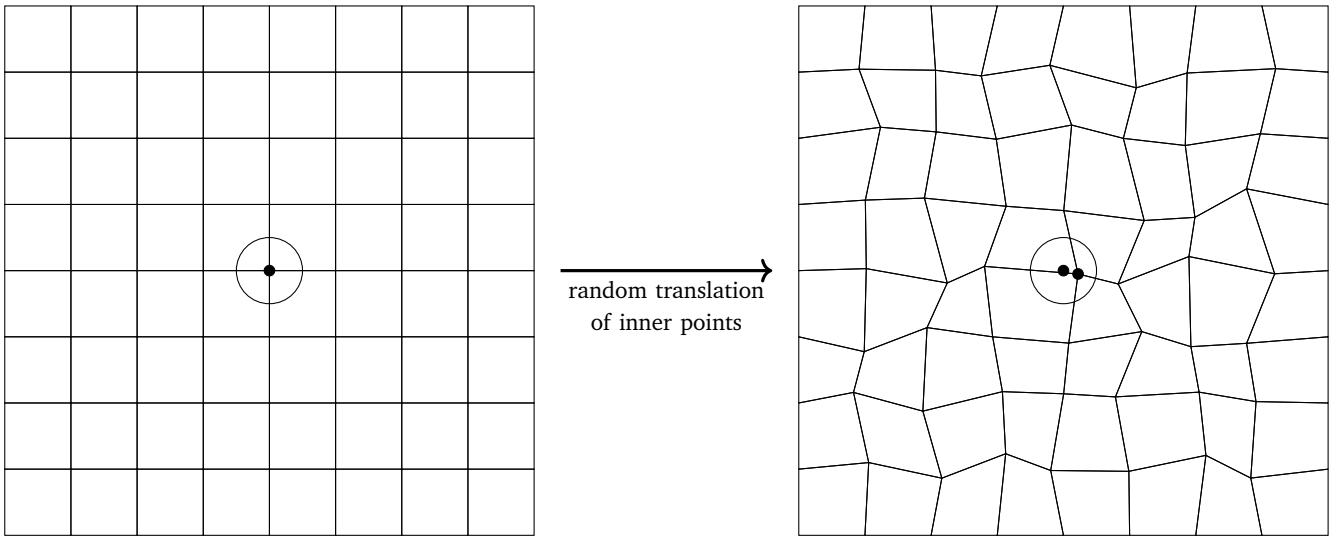


Figure 8.11: Skewing of an initially equidistant, structured and orthogonal grid via random movement of inner grid points within a small neighborhood of their original location. The neighborhood with a maximal diameter is indicated by a circle around one inner grid point

In order to measure the effect of different non-orthogonal corrections on the solution process, tests for different skewed grids were performed. For these tests, all correctors addressing grid non-orthogonality from section 3.3 have been implemented in the coupled solver program. The number of needed outer iterations on a $32 \times 32 \times 32$ grid to solve for the analytic solution presented in section 7.2 was measured. The grid skewness was parametrized with the relative

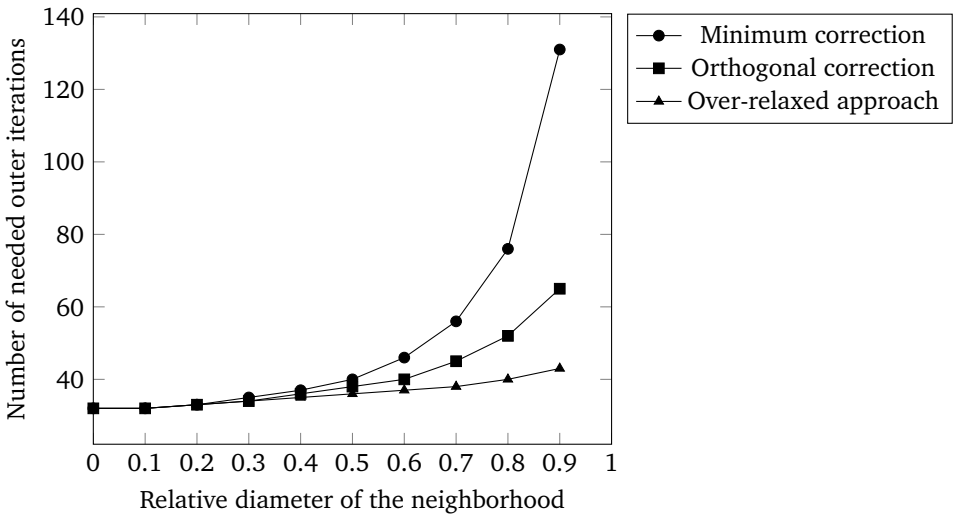


Figure 8.12: Number of needed outer iterations for different relative diameters of the neighborhood in which the movement of grid points takes place, parametrized by the orthogonal corrector

diameter α of the maximal neighborhood, which resulted in constraining the movement of grid points. $\alpha \in [0, 1)$, where $\alpha = 0$ corresponds to no movement of grid points at all and $\alpha = 1$ would move grid points up to $\frac{\Delta x}{2}$ away from their original location. The choice of $\alpha = 1$ is not permitted to maintain the grid's integrity. Such a choice would permit the grid generator to place two grid points at the same location. The calculations were terminated after obtaining a reduction of 10^{-14} of the relative initial residual.

The results presented in Figure 3.4 affirm the results presented in [**?**]. For small movements of grid points, the choice of corrector does not affect convergence. However, as movements of grid points increase, and consequently the non-orthogonality increases, the over-relaxed approach outperforms the other correctors with respect to the number of needed outer iterations. This fact also applies to performance with respect to wall-clock time since the computational effort of the application of each non-orthogonal corrector is approximately the same.

8.5 Achieving Ideal Load Balancing by Automatic Matrix Partitioning

The performance analyses conducted in section **??** relied on ideal load balancing and thus a load balancing efficiency of 100%. However, in practice, achieving and maintaining ideal load balancing throughout the solution process may be difficult. Local refinement strategies or an unfavorable grid blocking, i.e. the partition of the solution domain into grid blocks, may lead to an imbalance of computational load. This imbalance results in an inefficient utilization of employed hardware resources due to high idle times of processors. Furthermore, as equation (**??**) suggests, load imbalances result in an increase of wall-clock time to solve a given problem in parallel.

One possibility to partially address load imbalances is motivated by the fact that solution algorithms which solve partial differential equations spend a significant amount of their time solving linear algebraic systems. Based on the wall-clock time measurements for the solution of an analytic test case with 826891 unknowns, Figure **??** shows that approximately 48% of the wall-clock time of the segregated SIMPLE-solution algorithm, and up-to 89% of wall-clock time of the coupled solution algorithm are spent solving linear systems. The goal of this section is to present a simple method by which ideal load balancing, with respect to the linear equation solver, can be obtained using automatic matrix partitioning.

By using built-in PETSc subroutines, the rows of the corresponding matrices can be distributed automatically across all involved processes. With this technique, each process will retain the initially assigned variable data and calculate gradients, fluxes and matrix and right-hand side coefficients for the unknowns that correspond to the assigned portion of the numerical grid. Independently, the process gets assigned a portion of the matrix, a fact that may lead to the process assembling parts of the matrix that do not correspond to the coefficients which the process had calculated. Thus, additional communication becomes necessary during the matrix assembly step. This overhead may, however, be compensated by the more efficient utilization of linear equation solvers which, due to the ideal load balancing, will generate less processor idle times and make better use of the efficient implementations of the PETSc library.

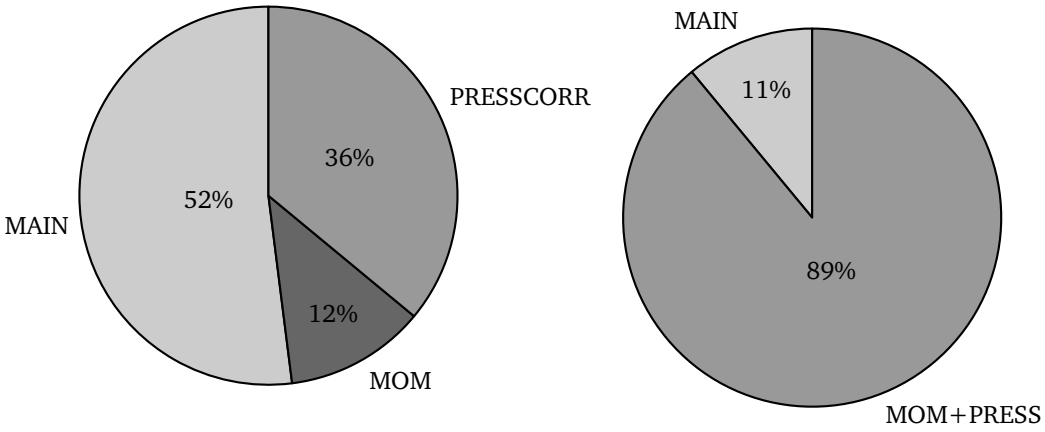


Figure 8.13: Distribution of wall-clock time spent to solve for an analytic solution with the segregated SIMPLE-algorithm (left) and the coupled solution algorithm (right) on one processor

The formula which PETSc uses to distribute vector or matrix objects containing N rows across m involved processes reads for the pth process

$$n(p) = \begin{cases} \lfloor N/m \rfloor + 1, & \text{if } N \mod m > p \\ \lfloor N/m \rfloor, & \text{else} \end{cases}, \tag{8.2}$$

where $p = 0, \ldots, m-1$ and $\lfloor \cdot \rfloor$ denotes the floor function. This function maps the possibly rational result of the integer division in (**??**) to the largest integer smaller than the result. It is evident that using equation (**??**) for load balancing leads to numbers of rows that differ by maximal one row across the involved processors.

In order to analyze the effect of the ideal matrix partitioning, a random number generator was used to create different grid resolutions for each one of the eight blocks of a block-structured grid, having a total of 826891 cells. Figure **??** shows the default distribution of matrix rows which correspond to the data which had been assigned to each process. Furthermore, Figure **??** compares the original data partitioning with the case using the automatic matrix partitioning. One can observe that, due to the simplicity of the load balancing technique, significant incoherences surge between the assigned grid and variable data and the assigned matrix rows. The rows are distributed in contiguous chunks, resulting in processes like process 0 or process 2 keeping high data coherence with the assigned matrix rows, while processes like process 3 or process 5 lack data coherence. Thus, more sophisticated distribution algorithms are necessary to maximize the retention rate of data for the individual processors maintaining the data coherence since a higher retention rate directly reduces the communication needed during matrix assembly. The utilization of advanced load distribution algorithms is especially recommendable for higher processor counts as the scattering of data and the corresponding matrix rows may significantly decrease performance.

The following paragraph analyzes the effects of the automatic matrix partitioning as a load balancing technique on solver performance, in particular, the following paragraph analyzes the effects on the absolute runtime and on the relative distribution of runtime among the phases in which the time was spent. The phases of the segregated solution algorithm are the MAIN phase, which accounts for the calculation of gradients, fluxes, matrix coefficients and the matrix assembly; the MOM phase, in which the linear systems of the momentum balances are solved; and the PRESSCORR phase, in which the linear systems are solved for the pressure-correction. Figure **??** shows that, for the presented test case, the automatic partitioning of the matrix not only decreases the needed wall-clock time but also increases the share of the MAIN phase of the total runtime. Matrix assembly during the MAIN phase is the only action of this phase which is affected by the new matrix distribution. Thus, initial load imbalances will still affect the calculation of gradients and other passages of the implementation which use global reduce operations and forced synchronization. One example for global reduce operations and forced synchronization is the calculation of the mean pressure to adjust the pressure level throughout the solution domain.

Depending on the parallel matrix layout, one processor might need to assemble parts of the matrix which do not reside in its address space. Due to the increased amount of communication necessary for the matrix assembly, relatively more time was spent in this phase compared to the test case using the default partitioning as Figure **??** shows. The absolute time difference is negligible, a fact that shows that the communication overhead during the matrix assembly step is more than compensated by the application of the ideally balanced matrix solver steps. All test calculations were performed using a single node of the MPI1 section of the HHLR. Thus, the presented results do not account for the additional communication performance degradation due to the effects of the inter-node interconnect.

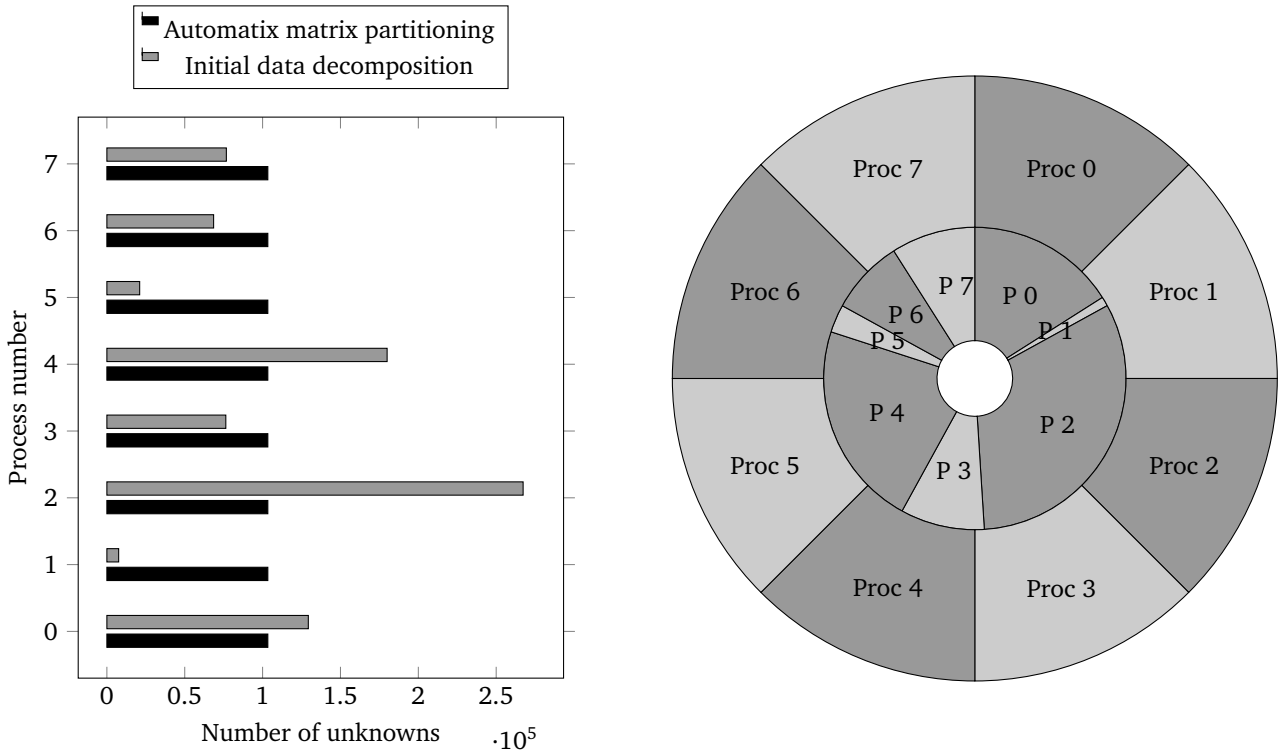


Figure 8.14: Comparison of the number of unknowns assigned to each process and the assigned rows of the matrix using automatic matrix partitioning for load balancing. The inner pie chart shows the initial distribution of the number of unknowns corresponding to the portion of the grid which has been assigned to a processor, whereas the outer pie chart shows the distribution of the automatically distributed matrix rows.

As Figure **??** shows, the beneficial effect of automatic matrix partitioning on solver performance is expected to increase significantly if applied to the matrix used in the coupled solution process. On the one hand, relatively more time is spent solving the linear system and, on the other hand, only one linear system is solved compared to the four linear systems solved in a segregated solution algorithm.

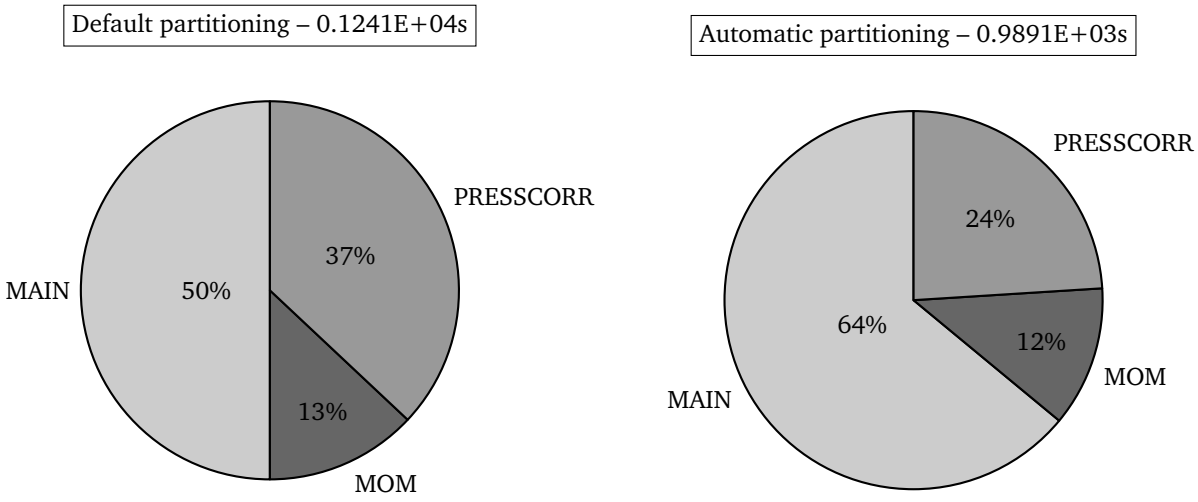


Figure 8.15: Distribution of wall-clock time for one and eight processes, solving for an analytic solution, using the default data partitioning for matrix assembly and automatic partitioning to balance the load. MAIN refers to the gradient calculations, to the flux calculations, and to matrix assembly. MOM refers to the solution of the linear systems for the momentum balance. PRESS refers to the solution of the linear systems for the pressure-correction.

9 Conclusion and Outlook

In the present thesis, a fully coupled solution algorithm for the Navier-Stokes equations with a finite-volume method was implemented. The method is designed to work with non-orthogonal, co-located, locally refined, block-structured grids and can handle hanging nodes resulting from non-matching block boundaries. The main difference of the implemented algorithm, compared to the commonly used segregated solution algorithms, was the implicit pressure-velocity coupling. The pressure-velocity coupling is a characterizing aspect of solution algorithms for Navier-Stokes equations with the potential to significantly reduce the time for computation. Furthermore, the solution algorithm was extended to solve for buoyancy-driven flows, by using an implicit Boussinesq approximation in the momentum balances to achieve velocity-to-temperature coupling and a semi-implicit Newton-Raphson linearization to realize implicit temperature-to-velocity/pressure coupling.

An existing solver framework, the CAFFA framework, which was extended during this thesis, was verified using a grid convergence study and a three-dimensional manufactured solution for the velocities, the pressure, and the temperature. Further studies on the segregated solution algorithm showed that a special modification of the widely used Rhie-Chow momentum interpolation scheme is necessary to calculate results that are independent of the under-relaxation factor and thus comparable with the results obtained with a fully coupled solution algorithm. The implementation of the algorithm considered the use of high-performance computers by using the PETSc library for parallelisation of the involved data structures and solving the resulting linear systems with solvers and preconditioners provided by the same toolkit.

A comparison study was conducted, dealing with the effect of implicit pressure-velocity coupling in the numerical solution process of the Navier-Stokes equations. Parallel performance measurements showed that, for the implementation of the segregated and the coupled solution algorithm, scalable program behavior is possible for high numbers of involved unknowns. Performance studies on flows through complex geometries demonstrated that the fully coupled solution algorithm outperforms the segregated solution algorithm. The analyses regarding the different degrees of velocity-to-temperature and temperature-to-velocity/pressure coupling showed that the implicit temperature-to-velocity/pressure coupling is the key aspect of maintaining high efficiency regarding the computation time for flow problems involving temperature transport and strong coupling between the velocities, the pressure, and the temperature.

Other studies performed in the present work showed that the over-relaxed approach, accounting for non-orthogonality of the numerical grid behaves superior to other commonly used non-orthogonal correctors, the orthogonal corrector, and the minimum corrector, when applied on heavily skewed numerical grids. Finally, a simple load balancing technique, relying on automatic matrix partitioning was presented, revealing the potential of achieving ideal load balancing for the involved linear solvers.

The present thesis revealed different starting points to examine the effects of coupled solution algorithms further in the context of finite-volume flow solvers. In the present thesis, only stationary problems were solved. Due to the moderate Rayleigh number for the heated cavity test case a stationary solution existed. By increasing the Rayleigh number the coupling between the velocities, the pressure, and the temperature gets stronger, and the flow will get instationary. A further investigation could examine if coupled solution methods maintain the higher efficiency compared to segregated solution algorithms when applied to instationary flows.

In this thesis, a temperature equation was successfully coupled to the modeling equations of a fluid. One straightforward application of the solver would deal with conjugate heat transfer problems. In practice there exist other similar transport equations that exhibit strong coupling to the velocities. Examples are the scalar transport of volume fractions, where multiphase flow problems are solved with Volume-of-Fluid methods, or turbulent flows, which, depending on the turbulence model, involve further scalar quantities that are strongly coupled to the velocities. Subsequent studies could examine if for the corresponding transport equations an implicit coupling method could be efficiently implemented.

All linear systems surging from the discretization process of the coupled system of partial differential equations were solved using black-box implementations of the available solvers in the PETSc library. As studies on finite-volume [**?**, **?**, **?**] and finite-element [**?**, **?**, **?**, **?**, **?**, **?**] discretizations show, a lot of information has not been used in the solution process of the linear system yet, a fact that imposes a barrier to performance. In order to exploit the coupling and the structure of the linear systems two different approaches can be considered. On approach features so-called *physics-based* preconditioners as the SIMPLE preconditioner or other Schur-type preconditioners introduced in [**?**, **?**]. This kind of preconditioners accelerates the solution of the linear systems by taking into account the matrix structure, as presented in figure 5.3. At the moment, further active research is needed to develop preconditioners for fully coupled systems surging from finite-volume discretizations. Another approach that additionally maintains high scalability of the solution algorithm of the resulting linear systems has been presented in [**?**, **?**]. In these references, algebraic multigrid methods are constructed which, unlike the GAMG preconditioner of PETSc, take into account the matrix structure.

As the studies in the present thesis demonstrated, segregated solution algorithms perform well compared to fully coupled solution algorithms for smaller numbers of unknowns. Furthermore due to the lower requirements of system memory, the problems that can be solved with segregated solution algorithms may, for very high numbers of unknowns, not be computable with fully coupled solution algorithms. Especially for flows involving temperature transport many other implementations of segregated solution algorithms exist [**?**, **?**], whose performance has not been compared with a coupled solution method.

Bibliography

[1] Acharya, S., Baliga, B. R., Karki, K., Murthy, J. Y., Prakash, C., and Vanka, S. P. Pressure-based finite-volume methods in computational fluid dynamics. *Journal of Heat Transfer 129*, 4 (Jan 2007), 407–424.

[2] Anderson, D. A., Tannehill, J. C., and Pletcher, R. H. *Computational Fluid Mechanics and Heat Transfer*. Hemisphere Publishing Corporation, Wahington, 1984.

[3] Aris, R. *Vectors, Tensors and the Basic Equations of Fluid Mechanics*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1962.

[4] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., and Zhang, H. PETSc users manual. Tech. Rep. ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014.

[5] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., and Zhang, H. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2014.

[6] Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F. Efficient management of parallelism in object oriented numerical software libraries. In *Modern Software Tools in Scientific Computing* (1997), E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds., Birkhäuser Press, pp. 163–202.

[7] Bonfiglioli, A., Campobasso, S., Carpentieri, B., and Bollhöfer, M. A parallel 3d unstructured implicit rans solver for compressible and incompressible cfd simulations. In *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, Eds., vol. 7204 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 313–322.

[8] Brown, J., Knepley, M. G., May, D. A., McInnes, L. C., and Smith, B. Composable linear solvers for multiphysics. In *Proceedings of the 2012 11th International Symposium on Parallel and Distributed Computing* (Washington, DC, USA, 2012), ISPDC '12, IEEE Computer Society, pp. 55–62.

[9] Chen, Z., and Przekwas, A. A coupled pressure-based computational method for incompressible/compressible flows. *Journal of Computational Physics 229*, 24 (2010), 9150 – 9165.

[10] Choi, S. K. Note on the use of momentum interpolation method for unsteady flows. *Numerical Heat Transfer, Part A: Applications 36*, 5 (1999), 545–550.

[11] Choi, S.-K., Kim, S.-O., Lee, C.-H., and Choi, H.-K. Use of the momentum interpolation method for flows with a large body force. *Numerical Heat Transfer, Part B: Fundamentals 43*, 3 (2003), 267–287.

[12] Christon, M. A., Gresho, P. M., and Sutton, S. B. Computational predictability of time-dependent natural convection flows in enclosures (including a benchmark solution). *International Journal for Numerical Methods in Fluids 40*, 8 (2002), 953–980.

[13] Darwish, M., Sraj, I., and Moukalled, F. A coupled finite volume solver for the solution of incompressible flows on unstructured grids. *Journal of Computational Physics 228*, 1 (2009), 180 – 201.

[14] Darwish, F. Moukalled, M. A unified formulation of the segregated class of algorithms for fluid flow at all speeds. *Numerical Heat Transfer, Part B: Fundamentals 37*, 1 (2000), 103–139.

[15] De Vahl Davis, G. Natural convection of air in a square cavity: A bench mark numerical solution. *International Journal for Numerical Methods in Fluids 3*, 3 (1983), 249–264.

[16] Elman, H., Howle, V. E., Shadid, J., Shuttleworth, R., and Tuminaro, R. A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible navier-stokes equations. *Journal of Computational Physics 227*, 3 (Jan. 2008), 1790–1808.

[17] Elman, H., Howle, V. E., Shadid, J., and Tuminaro, R. A parallel block multi-level preconditioner for the 3d incompressible navier-stokes equations. *Journal of Computational Physics 187* (2003), 504–523.

[18] Falgout, R., and Yang, U. hypre: A library of high performance preconditioners. In *Computational Science — ICCS 2002*, P. Sloot, A. Hoekstra, C. Tan, and J. Dongarra, Eds., vol. 2331 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2002, pp. 632–641.

[19] Falk, U., and Schäfer, M. A fully coupled finite volume solver for the solution of incompressible flows on locally refined non-matching block-structured grids. In *Adaptive Modeling and Simulation 2013* (Barcelona, Spain, June 2013), J. P. M. de Almeida, P. Diez, C. Tiago, and N. Parez, Eds., pp. 235–246.

[20] Ferziger, J. H., and Perić, M. *Numerische Strömungsmechanik*. Springer Verlag, Berlin, 2002.

[21] Galpin, P. F., and Raithby, G. D. Numerical solution of problems in incompressible fluid flow: Treatment of the temperature-velocity coupling. *Numerical Heat Transfer 10*, 2 (1986), 105–129.

[22] Gee, M., Siefert, C., Hu, J., Tuminaro, R., and Sala, M. ML 5.0 smoothed aggregation user's guide. Tech. Rep. SAND2006-2649, Sandia National Laboratories, 2006.

[23] Gray, D. D., and Giorgini, A. The validity of the boussinesq approximation for liquids and gases. *International Journal of Heat and Mass Transfer 19*, 5 (1976), 545 – 551.

[24] Gropp, W., Lusk, E., and Skjellum, A. *Using MPI: portable parallel programming with the message-passing interface*, 2. ed. The MIT Press, Cambridge, Massachussets, 1999.

[25] Gropp, W. D., Kaushik, D. K., Keyes, D. E., and Smith, B. F. High performance parallel implicit cfd. *Parallel Computing 27* (2000), 337–362.

[26] Hackbusch, W. *Theorie und Numerik elliptischer Differentialgleichungen mit Beispielen und Übungsaufgaben*. Teubner-Studienbücher Mathematik. Teubner, Stuttgart, 1996.

[27] Hager, G., and Wellein, G. *Introduction to High Performance Computing for Scientists and Engineers*. CRC, Boca Raton, 2011.

[28] Henderson, A. Paraview guide, a parallel visualization application, 2007.

[29] Issa, R. Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of Computational Physics 62*, 1 (1986), 40 – 65.

[30] Jasak, H. *Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows*. PhD thesis, Imperial College of Science, Technology and Medicine, Jun 1996.

[31] Karimian, S. M., and Straatman, A. G. Benchmarking of a 3d, unstructured, finite volume code of incompressible navier-stokes equation on a cluster of distributed-memory computers. *High Performance Computing Systems and Applications, Annual International Symposium on 0* (2005), 11–16.

[32] Klaij, C. M., and Vuik, C. Simple-type preconditioners for cell-centered, colocated finite volume discretization of incompressible reynolds-averaged navier-stokes equations. *International Journal for Numerical Methods in Fluids 71*, 7 (2013), 830–849.

[33] Kundu, P. K., Cohen, I. M., and Downling, D. R. *Fluid Mechanics*, 5 ed. Elsevier, 2012.

[34] Lange, C. F., Schäfer, M., and Durst, F. Local block refinement with a multigrid flow solver. *International Journal for Numerical Methods in Fluids 38*, 1 (2002), 21–41.

[35] Lilek, u., Muzaferija, S., Perić, M., and Seidl, V. An implicit finite-volume method using nonmatching blocks of structured grid. *Numerical Heat Transfer, Part B: Fundamentals 32*, 4 (1997), 385–401.

[36] Liu, X., Tao, W., and He, Y. A simple method for improving the simpler algorithm for numerical simulations of incompressible fluid flow and heat transfer problems. *Engineering Computations 22*, 8 (2005), 921–939.

[37] Majumdar, S. Role of underrelaxation in momentum interpolation for calculation of flow with nonstaggered grids. *Numerical Heat Transfer 13*, 1 (1988), 125–132.

[38] Mangani, L., Buchmayr, M., and Darwish, M. Development of a novel fully coupled solver in openfoam: Steady-state incompressible turbulent flows in rotational reference frames. *Numerical Heat Transfer, Part B: Fundamentals 66*, 6 (2014), 526–543.

[39] Maple. *version 18*. Waterloo Maple Inc. (Maplesoft), Waterloo, Ontario, 2014.

[40] MATLAB. *version 8.4.150421 (R2014b)*. The MathWorks Inc., Natick, Massachusetts, 2014.

[41] McCalpin, J. D. Stream: Sustainable memory bandwidth in high performance computers. Tech. rep., University of Virginia, Charlottesville, Virginia, 1991-2007. A continually updated technical report. `http://www.cs.virginia.edu/stream/`.

[42] McCalpin, J. D. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), 19–25.

[43] Mcinnes, L. C., Smith, B., Zhang, H., and Mills, R. T. Hierarchical krylov and nested krylov methods for extreme-scale computing. *Parallel Computing 40*, 1 (Jan. 2014), 17–31.

[44] Miller, T. F., and Schmidt, F. W. Use of a pressure-weighted interpolation method for the solution of the incompressible navier-stokes equations on a nonstaggered grid system. *Numerical Heat Transfer 14*, 2 (1988), 213–233.

[45] Muzaferija, S. *Adaptive finite volume method for flow predictions using unstructured meshes and multigrid approach*. PhD thesis, University of London, 1994.

[46] Oberkampf, W. L., and Trucano, T. G. Verification and validation in computational fluid dynamics. *Progress in Aerospace Sciences 38*, 3 (2002), 209 – 272.

[47] Oliveira, P. J., and Issa, R. I. An improved piso algorithm for the computation of buoyancy-driven flows. *Numerical Heat Transfer, Part B: Fundamentals 40*, 6 (2001), 473–493.

[48] Patankar, S., and Spalding, D. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal of Heat and Mass Transfer 15*, 10 (1972), 1787 – 1806.

[49] Patankar, S. V. *Numerical heat transfer and fluid flow*. McGraw - Hill, New York, 1980.

[50] Perić, M. Analysis of pressure-velocity coupling on nonorthogonal grids. *Numerical Heat Transfer 17* (Jan. 1990), 63–82.

[51] Pope, S. B. *Turbulent Flows*. Cambridge University Press, New York, 2000.

[52] R., H. M., and E., S. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards 49* (1952), 409.

[53] Rhie, C. M., and Chow, W. L. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal 21* (Nov. 1983), 1525–1532.

[54] Saad, Y. *Iterative methods for sparse linear systems*. SIAM, 2003.

[55] Saad, Y., and Schultz, M. H. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing 7*, 3 (July 1986), 856–869.

[56] Salari, K., and Knupp, P. Code verification by the method of manufactured solutions. Tech. Rep. SAND2000-1444, Sandia National Labs., Albuquerque, NM (US); Sandia National Labs., Livermore, CA (US), Jun 2000.

[57] Schäfer, M. *Numerik im Maschinenbau*. Springer Verlag, Berlin, 1999.

[58] Sheu, T. W. H., and Lin, R. K. Newton linearization of the incompressible navier–stokes equations. *International Journal for Numerical Methods in Fluids 44*, 3 (2004), 297–312.

[59] Silvester, D., Elman, H., Kay, D., and Wathen, A. Efficient preconditioning of the linearized navier–stokes equations for incompressible flow. *Journal of Computational and Applied Mathematics 128*, 1–2 (2001), 261 – 279. Numerical Analysis 2000. Vol. VII: Partial Differential Equations.

[60] Spurk, J. H., and Aksel, N. *Strömungslehre: Einführung in die Theorie der Strömungen*, 8. ed. Sringer Verlag, Berlin, 2010.

[61] Taylor, G. I., and Green, A. E. Mechanism of the production of small eddies from large ones. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences 158*, 895 (1937), 499–521.

[62] Turek, S., and Schmachtel, R. Fully coupled and operator-splitting approaches for natural convection flows in enclosures. *International Journal for Numerical Methods in Fluids 40*, 8 (2002), 1109–1119.

[63] Vakilipour, S., and Ormiston, S. J. A coupled pressure-based co-located finite-volume solution method for natural-convection flows. *Numerical Heat Transfer, Part B: Fundamentals 61*, 2 (2012), 91–115.

[64] Van Doormaal, J. P., and Raithby, G. D. Enhancements of the simple method for predicting incompressible fluid flows. *Numerical Heat Transfer 7*, 2 (1984), 147–163.

[65] Zhang, S., Zhao, X., and Bayyuk, S. Generalized formulations for the rhie–chow interpolation. *Journal of Computational Physics 258*, 0 (2014), 880 – 914.