

A Cut Cell Discontinuous Galerkin Method for Particulate Flows

Am Fachbereich Maschinenbau
an der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades
eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte

D i s s e r t a t i o n

von

Dennis Krause
aus Kreuztal

Berichterstatter:	Prof. Dr.-Ing. M. Oberlack
Mitberichterstatter:	Prof. Dr. techn. H. Egger
Tag der Einreichung:	18.12.2018
Tag der mündlichen Prüfung:	14.03.2019

Darmstadt, 2019

D17

Erklärung

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 18. Dezember 2018

Dennis Krause

Bitte zitieren Sie dieses Dokument als:

urn: urn:nbn:de:tuda-tuprints-92064

url: <https://tuprints.ulb.tu-darmstadt.de/id/eprint/9206>

Dieses Dokument wird bereitgestellt von tuprints, e-Publishing-Service der TU Darmstadt.

<https://tuprints.ulb.tu-darmstadt.de>

tuprints@ulb.tu-darmstadt.de



Die Veröffentlichung steht unter folgender Creative Commons Lizenz:

Namensnennung-Nicht kommerziell-Keine Bearbeitungen 4.0 International

<https://creativecommons.org/licenses/by-nc-nd/4.0>

Abstract

Today, there are still numerous phenomena of particulate flows in engineering and nature which are not fully understood. This results out of a lack of accurate and computationally efficient solvers in this field, especially when it comes to particles which deform to various shapes. The discontinuous Galerkin method can provide both, high accuracy and efficient parallelization due to its high order convergence and cell local formulations. Thus, it is a promising approach to better understand the underlying physics.

In this work a cut cell discontinuous Galerkin method is developed for particles with non-spherical shape. It is based on an immersed boundary approach to avoid costly remeshing. For the fluid part the Navier-Stokes equations and for the particle motion the Newton-Euler equations are solved. To connect both, a two-way coupling strategy is applied. It consists of the calculation of hydrodynamic forces acting on the particle and implying Dirichlet velocity boundary conditions on the particle surface for the fluid part. Furthermore, two collision models are implemented, both using a detection algorithm based on cut cells which also works for boundaries with arbitrary shape.

Various numerical experiments with increasing complexity show the high accuracy of the method by comparing the obtained results with literature. The investigations start from simple immersed boundary cases with non-moving domains to fully-coupled simulations of multiple particles falling in incompressible fluid. The method is extended to three dimensional problems to investigate a sphere at a Reynolds number of 700 as a proof of the possibility applying the proposed method to larger problems.

In order to obtain results for the three dimensional test in a reasonable time, the convergence of a newly implemented Newton-Krylov method with different preconditioning techniques is investigated. Further, the proposed method is optimized to run on multiple cores in parallel. For this, a performance analysis workflow for the open source framework BoSSS on high-performance computing (HPC) systems is presented. With this, a single-core performance analysis and tests focusing on the parallel efficiency of the current code are performed. In the end, the proposed method shows both, the ability of computing high accurate solutions for particulate flows and a good parallel scaling behavior on modern HPC systems.

Zusammenfassung

Heutzutage sind zahlreiche Phänomene von Partikelströmungen noch immer nicht vollständig verstanden. Dazu trägt ein Mangel von hochgenauen und effizienten Lösern bei, speziell für Partikel mit beliebiger Geometrie. Das diskontinuierliche Galerkin-Verfahren ermöglicht durch seine hohe Konvergenzordnung sowohl höchste Genauigkeit als auch effiziente Parallelisierbarkeit durch die zelllokalen Formulierungen. Daher ist sie ein vielversprechender Ansatz um die zugrundeliegende Physik zu untersuchen und besser zu verstehen.

In dieser Arbeit wird ein diskontinuierliches Galerkin-Verfahren mit geschnittenen Zellen für Partikel mit nicht-sphärischer Geometrie entwickelt. Dieses basiert auf dem Verfahren der eingebetteten Ränder um teure Gittergenerierung in jedem Zeitschritt zu vermeiden. Für das Fluid werden die Navier-Stokes-Gleichungen und für die Partikelbewegung die Newton-Euler-Gleichungen gelöst. Um diese zu verbinden, wird eine Kopplung in beide Richtungen verwendet. Diese besteht einerseits aus der Berechnung der hydrodynamischen Kräfte, die auf das Partikel wirken, und andererseits aus der Annahme von Dirichlet-Randbedingungen für die Partikelgeschwindigkeit auf der Partikeloberfläche. Des Weiteren werden zwei Kollisionsmodelle untersucht, die beide einen Detektionsalgorithmus basierend auf Zelnachbarschaftsverhältnissen für geschnittene Zellen verwenden. Dieser vorgestellte Algorithmus zeichnet sich durch seine hohe Flexibilität für verschiedenste Partikelgeometrien aus.

Zahlreiche numerische Experimente steigender Komplexität zeigen die hohe Genauigkeit des vorgestellten Verfahrens. Die Untersuchungen reichen von einfachen Rechnungen mit unbewegten Gebieten bis hin zu vollgekoppelten Simulationen mit mehreren Partikeln in inkompressiblem Fluid. Die Methode wird auf drei Dimensionen erweitert, um eine Kugelumströmung bei einer Reynolds-Zahl von 700 zu untersuchen und zeigt damit die Möglichkeit die vorgestellte Methode auf größere Probleme anzuwenden.

Um Resultate für dreidimensionale Rechnungen in einer vertretbaren Zeit zu erhalten, wird zunächst die Konvergenz einer neu implementierten Newton-Krylov Methode mit verschiedenen Vorkonditionierern untersucht. Des Weiteren wird das schrittweise Vorgehen innerhalb einer Rechenleistungsanalyse auf Hochleistungsrechnern für das Framework BoSSS ausgearbeitet. Mit Hilfe dieses Vorgehens werden verschiedene Untersuchungen, wie die Analyse des Betriebs auf einem Rechenkern sowie Tests der parallelen Effizienz durchgeführt. Schlussendlich zeigt die vorgestellte Methode beides, die Möglichkeit hoch genaue Lösungen für Partikelströmungen zu berechnen sowie eine gute parallele Skalierbarkeit auf modernen Hochleistungsrechnern.

Acknowledgments

At first I want to thank my supervisor Prof. Oberlack for giving me the opportunity to work on this research and the possibility to present my results at conferences all around the globe. Further, I thank Prof. Egger to be my second supervisor and giving me insights into the mathematical point of view during the whole time of this project.

Moreover, I want to acknowledge countless hours of discussions and support by Florian Kummer. The smooth start of this project was mainly the contribution of Björn Müller, who was always open for technical and organizational questions. Next, I want to thank my office mate Martin Smuda for countless discussions shaping my ideas to be realizable and for proofreading parts of this thesis. Further thanks goes to all (former) colleagues supporting me with their knowledge and/or proofreading parts of this thesis. Those are in alphabetical order: Benjamin Deußen, Markus Geisenhofer, Anne Kikker, Stephan Krämer-Eis, Thomas Utz and Jens Weber. In addition, I want to thank Benedikt Niessen for proofreading introduction and conclusion from a non-expert perspective.

I also want to thank my friends from Darmstadt for many distractions and numerous hours of laughter and joy, which is an important contribution to the outcome of this thesis as well.

Finally, special thanks go to my parents who gave me the opportunity to realize my plans, both in Siegen and especially after moving to Darmstadt. This work would not have been possible without their continuous support. Last but not least I want to thank my fiancée Samira Zoghiami for going this road with me and constantly supporting me in all meaningful and meaningless plans during our relationship.

This work was supported by the 'Excellence Initiative' of the German Federal and State Governments and the Graduate School of Computational Engineering at Technische Universität Darmstadt. This research was in part also supported through computational resources provided by the Lichtenberg high performance computer at Technische Universität Darmstadt.

Contents

List of Figures	xiii
List of Tables	xv
Nomenclature	xvii
Abbreviations	xix
1 Introduction	1
1.1 Goal of this work	2
1.2 The open source framework BoSSS	3
1.3 Outline of this thesis	4
2 Mathematical model	5
2.1 Fluid flow	5
2.2 Particle movement	6
2.3 Coupling	7
2.4 Modifications to incorporate collisions	8
3 State of the art	11
3.1 Literature review	11
3.2 Splitting scheme	12
3.3 Treatment of collisions	13
3.3.1 Collision models	13
3.3.2 Efficient collision detection	17
3.4 Immersed boundary methods	21
3.4.1 Coupling forces at volume	21
3.4.2 Coupling forces at surface	24
4 Cut cell discontinuous Galerkin (DG) method for particulate flows	29
4.1 Splitting scheme	29
4.2 Discontinuous Galerkin method for flow problem	30
4.3 Cut cell collision model	34
4.3.1 Momentum conservation and repulsive force	35
4.3.2 Collision detection based on cut cells	36
4.4 Integration and agglomeration on cut cells	38
4.5 Conclusion	39

5	Numerical results	41
5.1	Flow around particles with circular shape	42
5.1.1	Flow around a fixed cylinder	42
5.1.2	Flow around a transversally oscillating cylinder	45
5.1.3	One rotating particle in a shear flow	48
5.1.4	Single particle falling in an incompressible fluid	51
5.1.5	Draft, kissing and tumbling of two circular particles	55
5.2	Particles with non-circular shape	58
5.2.1	Elliptic disk falling with various initial angle	58
5.2.2	Dry collisions of multiple particles	63
5.2.3	Five particles of different shape falling in fluid	64
5.3	Extension to three dimensions	66
5.3.1	Stationary flow around a sphere	66
5.3.2	Transient flow around a sphere	67
5.4	Conclusion	67
6	Performance analysis and profiling	69
6.1	State of the art	69
6.1.1	Equation system	70
6.1.2	High-performance computing	70
6.1.3	Software performance testing	74
6.2	Solving the coupled nonlinear equation system	75
6.2.1	Nonlinear solver	76
6.2.2	Linear solver	77
6.2.3	Choice of preconditioning technique	78
6.2.3.1	Schur methods	79
6.2.3.2	Schwarz methods	80
6.3	Precondition benchmarks	83
6.3.1	Boundary fitted	84
6.3.2	Immersed boundary	86
6.4	Performance analysis	90
6.4.1	Proposed workflow	90
6.4.2	Single core performance	92
6.4.3	Parallel performance tuning	94
6.4.4	Current bottlenecks	99
6.5	Conclusion	102
7	Conclusion and outlook	105
A	Appendix	119
A.1	A localized operator preconditioner	119
	Curriculum vitae	121

List of Figures

1.1	Target region of the <i>BoSSS</i> -framework.	3
2.1	Schematic computational domain with fluid and particle part.	6
2.2	Eccentric collision between an ellipse i and a circle j	9
3.1	Force depending on distance in the model of Glowinski et al. (2001).	14
3.2	Force depending on distance in the model of Wan and Turek (2006).	15
3.3	Particle quantities used by Maury (1997)	16
3.4	Back-face culling technique for two circular bodies.	20
3.5	Shape of a particle taken from Duchanoy and Jongen (2003).	25
3.6	Shell with equidistant force points taken from Uhlmann (2005).	26
4.1	Solver process scheme using a explicitly coupled method between fluid and particle part.	31
4.2	Eccentric collision zoomed in point of collision.	35
4.3	Collision detection based on cut cells for arbitrary geometry.	37
4.4	Determination of interface points inside a cut cell.	38
4.5	$K_{j+1,c}$ is agglomerated to its nearest neighbor $K_{j,c}$	40
5.1	Overview of numerical results.	41
5.2	Computational domain taken from Schäfer et al. (1996).	42
5.3	Results of different penalty factors for $Re=20$	44
5.4	Results of different penalty factors for $Re=100$	45
5.5	Background mesh for calculation with $k = 3$	47
5.6	Drag hysteresis for one cylinder oscillation.	48
5.7	Velocity magnitude for different radii.	49
5.8	Angular velocity of particle with different radii over time.	50
5.9	hp-convergence for velocity and pressure. For each polynomial degree the ideal slope is shown.	50
5.10	Background mesh for $k = 3$	52
5.11	Results of a single disk falling in incompressible fluid.	53
5.12	t-convergence of Lie-splitting.	53
5.13	Velocity magnitude at different times t	56
5.14	Comparison of collision models with $k = 2$	57
5.15	Behavior under h -refinement for momentum conservation model with $k = 2$	57
5.16	Different starting angles of an elliptic disk falling in incompressible fluid.	59
5.17	Results of an elliptic disk falling with $\theta_0 = 0^\circ$	60
5.18	Results of an elliptic disk falling with $\theta_0 = 45^\circ$	61

5.19	Results of an elliptic disk with $\theta_0 = 90^\circ$	62
5.20	View on the collision test at different points in time.	64
5.21	Velocity magnitude of particles with different shape.	65
5.22	Velocity magnitude and streamlines of a sphere flow with $Re=100$	66
6.1	Shared memory system, in which each CPU can contain several cores.	72
6.2	Distributed memory system.	73
6.3	Hybrid memory system containing SMP nodes.	73
6.4	Block-Jacobi scheme.	81
6.5	Overlapping domain decomposition.	82
6.6	Picard-GMRES convergence for setting 3 with immersed sphere.	89
6.7	Newton-GMRES convergence for setting 3 with immersed sphere.	90
6.8	Proposed performance tuning cycle for usage in Subsection 6.4.3.	91
6.9	Increasing DoF for direct and iterative solver.	94
6.10	Sequential and parallel parts of equation solve process.	96
6.11	Strong scaling results for solver parts <u>before</u> tuning.	97
6.12	Strong scaling results for different cut cell distributions among cores.	98
6.13	Overall parallel efficiency <u>after</u> tuning.	99
6.14	Percent of time spent in methods for 3D channel.	100
6.15	Percent of time spent in methods for 3D sphere.	101
6.16	Percent of time spent in methods for falling ellipse 45°	102

List of Tables

5.1	Results for steady flow around a fixed cylinder at $Re=20$	43
5.2	Results for unsteady flow around a fixed cylinder at $Re=100$	44
5.3	Results for unsteady flow around an oscillating cylinder.	47
5.4	Results of a particle in shear flow.	49
5.5	Experimental order of h-convergence.	51
5.6	Results single particle falling in incompressible fluid.	51
5.7	Results splitting vs. moving interface in a refined setting.	54
5.8	Dependency of moving domain approaches on time accuracy.	55
5.9	Points in time where collision effects occur for the calculation of the refined mesh in Figure 5.15.	58
5.10	Setting for falling elliptic disk.	59
5.11	Initial particle settings for collision test.	63
5.12	Initial particle settings for five particles falling.	65
5.13	Results of a sphere flow at $Re=100$	67
5.14	Results of a sphere flow at $Re=700$	67
6.1	Computational settings for preconditioning tests with channel flow and immersed cylinder/sphere meshes.	84
6.2	GMRES iterations for setting 1 with channel flow.	85
6.3	GMRES iterations for setting 2 with channel flow.	85
6.4	GMRES iterations for setting 3 with channel flow.	86
6.5	GMRES iterations for setting 1 with immersed cylinder.	87
6.6	GMRES iterations for setting 2 with immersed cylinder.	87
6.7	GMRES iterations for setting 3 with immersed sphere.	88
6.8	GMRES(m) iterations for setting 3 with immersed sphere (IB).	88
6.9	Time in 10^4 seconds for 3D Channel Flow with $\mu = 0.01$	93
6.10	Time in 10^4 seconds for 3D Sphere Flow with $\mu = 0.01$	93
6.11	Computational setting 4 extending setting 3.	96
6.12	Configurations for current bottleneck measurements.	99

Nomenclature

Fluid Symbols

μ_f	Dynamic viscosity
ν_f	Kinematic viscosity
Ω_f	Fluid domain
p	Pressure
$\partial\Omega_f$	Fluid boundary
ρ_f	Fluid density
\vec{u}	Fluid velocity vector
\vec{x}	Spatial coordinate vector

Particle Symbols

a_j	Normal distance between line of collision and center of mass
\vec{C}_j	Point of collision
e	Coefficient of restitution
\vec{F}_j	Hydrodynamical force
Γ_j	Particle boundary
I_j	Moment of inertia tensor
M_j	Particle mass
\vec{n}_c	Collision normal
$\vec{\omega}_j$	Angular velocity
Ω_j	Particle domain
ρ_j	Particle density
\vec{t}_c	Collision tangential
\vec{T}_j	Hydrodynamical torque
$\vec{\theta}_j$	Particle angle
\vec{U}_j	Particle velocity vector
\vec{X}_j	Center of mass

Other Symbols

$\mathbf{1}$	Identity matrix
$\mathbf{1}_{\Omega_f(t)}$	Characteristic function
d_{ij}	Particle-particle or particle-wall distance
D	Number of spatial dimensions
Δt	Size of a time step
μ	Penalty safety factor
η	Penalty parameter for SIP
G_{K_h}	Geometrical factor depending on the surface to volume ratio of a particular cut cell
h	Characteristic mesh size
h_{min}	Minimum mesh size
J	Jacobian matrix
k	Polynomial degree for discontinuous Galerkin method
K_j	Cell
$K_{j,c}$	Cut cell
$\mathcal{K}_h(t)$	Computational grid
\mathcal{K}_h^B	Computational background grid
L	Length scale
λ	Lax-Friedrichs parameter
Ω	Problem domain
Ω_h	Discrete problem domain
T	Total time
t	Time
V_k	DG space

Abbreviations

BoSSS	Bounded Support Spectral Solver
DoF	degrees of freedom
BDF	backward differencing formula
BDF-2	backward differencing formula of second order
IBM	immersed boundary method
FSI	fluid-structure interaction
DG	discontinuous Galerkin
XFEM	extended finite element method
XDG	extended discontinuous Galerkin
FEM	finite element method
FVM	finite volume method
ALE	arbitrary Lagrangian Eulerian
NSE	Navier-Stokes equations
GMRES	generalized minimal residual method
NEE	Newton-Euler equations
HMF	hierachical moment-fitting
PDE	partial differential equation
ODE	ordinary differential equation
CFD	computational fluid dynamics
SIP	symmetric interior penalty
RHS	right-hand side
EOC	experimental order of convergence
HPC	high-performance computing
DNS	direct numerical simulation
UMA	uniform memory access
SMP	symmetric multi-processing

NUMA non-uniform memory access

COMA cache-only memory architecture

OpenMP Open multi-processing

MPI message passing interface

API application programming interface

JSON JavaScript Object Notation

GUI graphical user interface

Re Reynolds number

St Strouhal number

SIMPLE semi-implicit method for pressure linked equations

1 Introduction

During the past years the importance of numerical methods for engineering applications has been under steady growth. Today, it is the third important pillar of understanding physics, besides theory and experiment. In industry numerical simulations are performed for various applications, like structural mechanics and thermodynamics. Probably the most important one is computational fluid dynamics (CFD), which is used, for example, for the design of wind turbines, combustion engines, and car aerodynamics. The current state of the art methods for CFD in industry are the finite volume method (FVM) of first, and the finite element method (FEM) of up to third order.

The discontinuous Galerkin (DG) method combines advantages of both, FVM and FEM. DG is a method for arbitrary order discretization which uses discontinuous polynomial for approximations. The cells are coupled through the usage of fluxes over cell boundaries which leads to a natural conservation property. Additionally, it is flexible for local refinement, of both h and k in the area of interest. A major drawback of DG are the increasing amount of degrees of freedom (DoF) in comparison with FEM and the larger stencils of the system matrices. However, methods of high-order are driven to be applicable in industrial computations every year more. At the moment high-order methods in industry are applied at most for problems with single-phase fluid flow instead of multiphase or particulate flows due to their high complexity.

Nonetheless, applications and phenomena of particulate flows are present all around the daily life. Researchers and engineers are interested to simulate the behavior of such flows because most of the time experiments are only possible with high costs or are not feasible at all. In nature, the motion of wind driven sand dunes or the sedimentation in ocean or river beds can be simulated using particulate CFD. In industrial applications the field of chemical and process engineering is usually called.

For all of the aforementioned applications, particles can be assumed to be spherical and to be present in large amounts. However, for lots of possible applications those assumptions cannot be made. Thus, other applications with fewer particles but high requirements to accuracy come into mind. The simulation of blood flow is a challenging task in life science in order to design stents or artificial heart valves. Here, red blood cells deform depending on the amount of oxygen they carry and cannot be assumed spherical, especially in small vessels. Other possible applications are the simulation of active particles like bacteria with self-driven motion in a surrounding fluid. Further, the simulation of microfluidic structures like yeast-cells on a microchip is under growing interest for researches in the field of electric engineering and biology. In detail, the design of such labs-on-a-chip trapping geometries for single cells require high-accurate solvers.

As a result, particulate flows are of high interest in science and technology with various applications from large to small scales with high accuracy requirements. On the macro- and microscopic scale, the physics behind particulate flows are not fully understood and still under active research in science and industry. Those simulations are highly complex, because particle position as well as their velocities are unknown. Even nowadays, with the computing power of high-performance computing (HPC) clusters, keeping the solver to be efficient and of high accuracy at the same time is still an ongoing challenge. Thus, there is still a lack of high-performance solvers for particulate flows to resolve all the physical effects in a reasonable amount of time.

Popular methods are the work of Uhlmann (2005) using first order FVM and the work of Wan and Turek (2006) who use a second order FEM discretization. In addition, both use a smooth interface representation of the particle surface and all particles are mainly assumed to be of circular shape. To summarize, the development of a high-order method for particulate flows with a sharp-interface representation of arbitrary shapes is still an very active field in research.

1.1 Goal of this work

The work of this thesis aims towards a high-order numerical method for particulate flows for arbitrary shaped particles and evaluate computational efficiency and possible bottlenecks subsequently. Thus, the core concept of this thesis is to develop a high-order method for the simulation of flows containing particles with arbitrary shape using immersed boundaries to avoid computationally expensive re-meshing. The DG method is used because of its abilities to be very suitable for adapting spatial accuracy in regions of interest.

For numerical discretization high-order test and ansatz functions are considered together with a sharp interface representation using a characteristic function to describe the particle surface. It is shown in several publications (Chapelier et al., 2014; de Wiart et al., 2015), that a similar low order spatial accuracy can be obtained by saving DoF due to high order test and ansatz functions. With the help of accurate quadrature on cells cut by the interface, the method proofs to be of high-order accuracy in space. Using a conservative collision model together with a collision detection for arbitrary shaped particles enables the method to be applied for various fluid-particle configurations.

In a next step, the foundation for future engineering applications is built. For this, a performance tuning cycle tailored to the specific code framework is proposed. In addition, the solution of nonlinear and linear problems arising in every timestep is investigated in order to highlight possible performance bottlenecks. In the end, the proposed method is able to run on a large number of cores in parallel and achieves high scalability.

1.2 The open source framework BoSSS

The cut cell DG method presented in this thesis is implemented into the open-source framework *Bounded Support Spectral Solver (BoSSS)*. In addition, the proposed performance analysis workflow is especially tailored to the framework and therefore results of all measurements obtained are those based on the performance of *BoSSS*.

The development of the framework was started in 2008 at the Chair of Fluid Dynamics (FDY) and is based on a DG numerical discretization in order to treat physical problems with a higher-order approach, see Kummer (2012). It is implemented using the C#-language to be platform independent, so it can be used on Windows, Linux and MacOS.

BoSSS was initiated to overcome the gap between simple matlab prototypes and very specific codes in high-performance computing, see Figure 1.1. In engineering, matlab codes are often used but have the well-known drawback of a limited performance. Writing code tailored to a specific application can yield good performance results but is often not feasible for engineers. *BoSSS* aims in filling this gap of delivering more performance than a matlab code but with providing a framework for more easy usage than a fully own developed high-performance code.

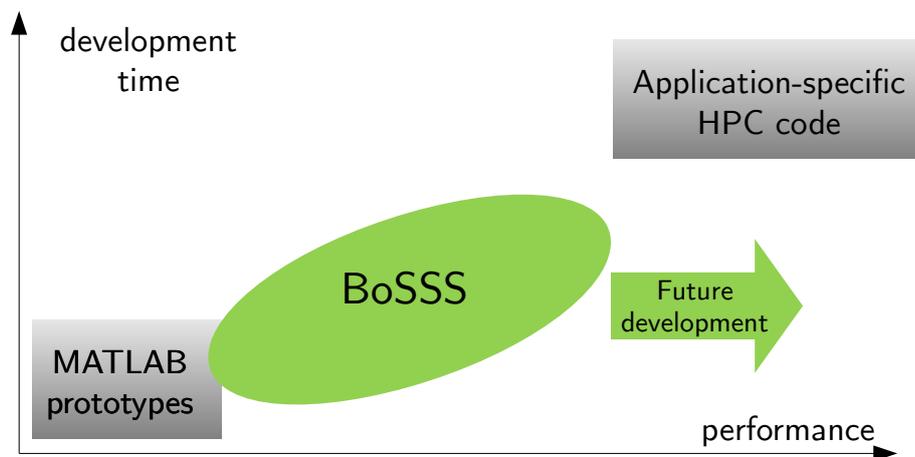


Figure 1.1: Target region of the *BoSSS*-framework.

The *BoSSS*-framework has already been the basis of many dissertations and publications. It was applied for compressible flow problems (Müller, 2014; Müller et al., 2016), low Mach number flows (Klein et al., 2016) as well as for incompressible flows (Klein et al., 2012, 2015). Further progress was made in terms of multiphase flow applications using cut cells (Kummer, 2016), which is related to the presented work. *BoSSS* was also used to develop general methods for cut cell integration (Müller et al., 2013), time discretization for moving domains (Kummer et al., 2017) and level-set movement (Utz et al., 2017; Utz and Kummer, 2018). This list only denotes related work to this thesis.

Note that the code is under active development at FDY at TU Darmstadt. The current version of the code can be found on the website <https://github.com/>

`fdydarmstadt/bosss`. On this page, the latest documentation of the code with further information can also be found.

1.3 Outline of this thesis

The following chapters of this thesis are organized as follows:

In Chapter 2 the basic mathematical model for the numerical method is given. It contains the Navier-Stokes equations (NSE) for incompressible fluid flow, the Newton-Euler equations (NEE) for modeling the movement of rigid particles, the coupling of those two models and shortly introduces the modifications which have to be made to incorporate collisions which conserves momentum and kinetic energy.

In Chapter 3 commonly used methods in literature for immersed boundary discretizations with moving domains are highlighted together with treatment of collisions published in this context. In the end of the chapter are more detailed few is given on immersed boundary methods and the coupling approach between fluid and particle equations being either the applications of forces at the volume or at the surface of the given particle.

The numerical method developed in this thesis is presented in Chapter 4. After that the proposed method is presented in detail for the flow region containing spatial and temporal discretization two cut cell collision models are introduced. Lastly, some short notes on key aspects of the method being accurate quadrature and cell-agglomeration are given.

Chapter 5 contains various results of numerical experiments used for validating the presented method. This is done by a comparison with common methods in literature. The experiments are split into particles with circular shape, particles with non-spherical shape, and the extension of the method to three dimensions. The complexity of the numerical experiments in this chapter increases from beginning to end successively, testing more and more features of the presented method.

An investigation with regard to computational efficiency is presented in Chapter 6. First, the current standards of HPC and performance measurements are introduced. After that the extension to an iterative solver for the coupled nonlinear equation system is made including the test of different preconditioning techniques. An important outcome is the proposed performance tuning workflow for the overall *BoSSS*-framework. The current solver performance is measured for single-core and parallel runs. Further, current bottlenecks are pointed out in the end.

Finally, in Chapter 7 a conclusion of this work is given, which will refer the main outcomes of this thesis to the previous defined goal of this work. Additionally, an outlook for future extensions and a view on the overall long term goal will be presented.

2 Mathematical model

In the beginning it is important to state that, according to Sigurgeirsson et al. (2001), there are three different types of particulate flows including collisions with increasing difficulties: (I) In Billiards the particles move with constant velocities and straight lines between collisions with no presence of fluid flow. In Particle laden flows (II) there is only coupling from fluid onto the particles and not vice-versa. This work mainly focuses on type (III), coupled particle-flow, in which fluid and particle motion are coupled in both directions.

In the following, the mathematical modeling for type (III) flows will be presented including the coupling between those fluid and particle part. Furthermore, the modification to incorporate rigid body collisions will be explained. This chapter will only focus on the mathematical aspects, postponing discussions about numerical details in the discrete setting to the next chapter.

2.1 Fluid flow¹

Here, the description of the fluid flow modeling using the NSE will be given. For more detailed information see e.g. Glowinski et al. (1999) or Wan and Turek (2006). For introducing the immersed boundary method (IBM) a disjoint partitioning of the computational domain $\Omega \subset \mathbb{R}^D$ is defined:

$$\Omega_f(t) = \Omega \setminus \overline{\Omega}_p(t) \quad (2.1)$$

and

$$\Omega_p(t) = \cup_j \Omega_j(t) \quad (2.2)$$

with $\Omega_j(t)$ denoting the domain occupied by the j -th particle at time t and $\Omega_f(t)$ denoting the remaining fluid domain. We further defined Dirichlet- and Neumann boundaries

$$\Gamma_D \cup \Gamma_N = \partial\Omega_f(t) \setminus \partial\Omega_p(t) \quad (2.3)$$

and particle boundaries

$$\Gamma_j(t) = \partial\Omega_j(t). \quad (2.4)$$

A schematic Figure of the computational domain can be seen in Figure 2.1. Therefore, physical parameters like densities will be split into fluid part, e.g. ρ_f in Ω_f , and particle part, e.g. ρ_j in Ω_j .

The immersed boundary solver will be used to calculate incompressible flows in interaction with arbitrary shaped bodies.

¹Modified version of (Krause and Kummer (2017), Section 2.1)

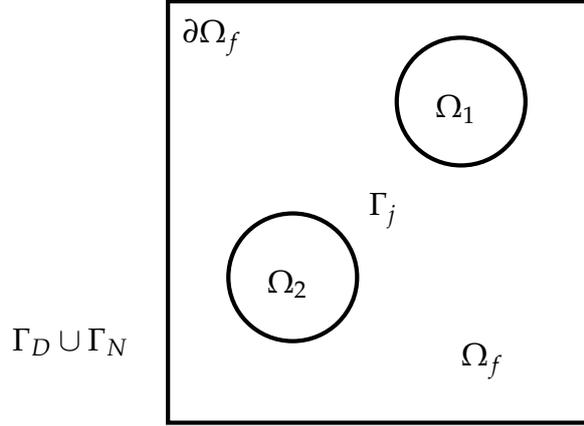


Figure 2.1: Schematic computational domain with fluid and particle part.

The flow in the fluid domain is described by the unsteady NSE in the fluid region

$$\rho_f \left(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} \right) - \nabla \cdot \boldsymbol{\sigma} = \vec{f} \quad \forall t \in (0, T) \quad \text{in } \Omega_f(t) \quad (2.5a)$$

where u is the velocity field, ρ_f is the fluid density, and f is the sum of external forces. We further assume that the fluid is Newtonian and therefore define the total stress tensor as $\boldsymbol{\sigma}$, which is defined as follows:

$$\boldsymbol{\sigma} = -p\mathbf{1} + \mu_f [\nabla \vec{u} + (\nabla \vec{u})^T], \quad (2.5b)$$

where p is the pressure, μ_f the viscosity of the fluid, and $\mathbf{1}$ is the identity tensor.

The incompressibility condition for the flow is encoded in

$$\nabla \cdot \vec{u} = 0 \quad \forall t \in (0, T) \quad \text{in } \Omega_f(t). \quad (2.5c)$$

The system is complemented by the following initial and boundary conditions:

$$\vec{u}(\vec{x}, 0) = \vec{u}_0(\vec{x}) \quad \forall \vec{x} \in \Omega_f(0) \quad \text{with} \quad \nabla \cdot \vec{u}_0 = 0 \quad (2.5d)$$

$$\vec{u} = \vec{u}_D \text{ on } \Gamma_D, \quad -\boldsymbol{\sigma} \cdot \vec{n}_{\Gamma_N} = 0 \text{ on } \Gamma_N \text{ and } \vec{u} = \vec{u}_j \text{ on } \Gamma_j. \quad (2.5e)$$

In the equations above \vec{u} is the velocity vector, p the pressure and \vec{u}_j the rigid body velocity imposed at the immersed boundary Γ_j of the particular particle. The fluid density is denoted by ρ_f , while $\mu_f = \rho_f \cdot \nu_f$ is the dynamic viscosity of the fluid. Furthermore, volume forces acting on the fluid are described by the force vector \vec{f} .

2.2 Particle movement²

The rigid particle movement is driven by forces exerted by the fluid. Detailed information can be found in the related work of Glowinski et al. (1999), Wan and Turek (2006)

²Modified version of (Krause and Kummer (2017), Section 2.2)

or in literature covering kinematics in general like Gross et al. (2008). In the following, capital letters for Lagrangian coordinates are used to prevent any confusion.

Remember that $\Omega_j(t) = \{X(t) : X(0) \in \Omega_j(0)\}$ is the area occupied by the j -th particle at time t and the particle movement is assumed to be rigid. Thus, the movement of a particle can be expressed due to the position of the center of mass and his rotational angle. For initialization, a particle is defined by its position $\vec{X}_j(0)$, its angle $\vec{\theta}_j(0)$, its mass M_j and the moment of inertia tensor \mathbf{I}_j .

With this and given translational and rotational velocities U_j and ω_j the center of mass $\vec{X}_j(t)$ of the particle and its angle $\vec{\theta}_j(t)$ is obtained by time integration of the kinematic relations

$$\frac{d\vec{X}_j}{dt} = \vec{U}_j \quad (2.6a)$$

and

$$\frac{d\vec{\theta}_j}{dt} = \vec{\omega}_j. \quad (2.6b)$$

The particle velocities can be determined by solving the NEE in time:

$$M_j \frac{d\vec{U}_j}{dt} = M_j \vec{f} + \vec{F}_j \quad (2.7a)$$

and

$$\mathbf{I}_j \frac{d\vec{\omega}_j}{dt} + \vec{\omega}_j \times \mathbf{I}_j \vec{\omega}_j = \vec{T}_j, \quad (2.7b)$$

where \vec{f} are volume forces like gravity and \vec{F}_j the hydrodynamic forces acting on the body. For the rotational equation T_j the hydrodynamic torque. Please note that the term $\vec{\omega}_j \times \mathbf{I}_j \vec{\omega}_j$ vanishes in a two-dimensional setting. Both, \vec{F}_j and T_j are assumed to be given. Details about the coupling will be postponed to the next section.

2.3 Coupling³

The hydrodynamic forces and torque are calculated by

$$\vec{F}_j(t) = \int_{\Gamma_j} \boldsymbol{\sigma}(\vec{x}, t) \cdot \vec{n}(\vec{x}, t) \, ds(\vec{x}), \quad (2.8a)$$

$$\vec{T}_j(t) = \int_{\Gamma_j} (\vec{x} - \vec{X}_j) \times (\boldsymbol{\sigma}(\vec{x}, t) \cdot \vec{n}(\vec{x}, t)) \, ds(\vec{x}). \quad (2.8b)$$

Equations (2.8) contain the surface integration of the total stress tensor (2.5b) along the body surface. The term $(\vec{x} - \vec{X}_j)$ denotes the distance between a point on the particle surface and its center of mass. Note that for circular shaped bodies this is just the radius r .

³Modified version of (Krause and Kummer (2017), Section 2.2)

For the solution of the fluid flow problem, the particle domain $\Omega_p(t) = \cap_j \Omega_j(t)$ has to be determined. In terms of rigid particles the representation of the particle boundary Γ_j can be reconstructed out of the solution of (2.6) for the center of mass \vec{X}_j and the angle $\vec{\omega}_j$.

At the interface between fluid and body Γ_j , no-slip boundary conditions are enforced. Hence, the particle velocity at the surface described by

$$\vec{u}_j(\vec{x}, t) = \vec{U}_j + \vec{\omega}_j \times (\vec{x} - \vec{X}_j) \quad \text{on } \Gamma_j \quad (2.9)$$

is imposed at the interface, as it can be seen in (2.5e). This technique is well known for fluid-structure interaction (FSI) problems, see, e.g. Mittal and Iaccarino (2005); Hou et al. (2012); Peskin (2002).

2.4 Modifications to incorporate collisions

First it will be focused on collisions in general if particles move freely following the NEE out of Section 2.2. Collisions of rigid bodies are based on the conservation of momentum at the time of collision. This section will include a model for bodies with arbitrary shape. Therefore, the model contains eccentric parts because during collision the centers of mass do not have to be aligned, like it is the case for spherical particles. This leads the model to contain parts for rotational motion as well. The collision of rigid bodies is well known and can be found for example in Gross et al. (2008).

During the collision modeling, the local coordinate system is rotated along to the collision normals between two particles. Then, the translational velocities of both particles are transformed in (2.10) to a normal U_n and a tangential U_t component in the local coordinate system.

$$U_{in} = \vec{U}_i \cdot \vec{n}_c, \quad U_{it} = \vec{U}_i \cdot \vec{t}_c \quad (2.10a)$$

$$U_{jn} = \vec{U}_j \cdot \vec{n}_c, \quad U_{jt} = \vec{U}_j \cdot \vec{t}_c \quad (2.10b)$$

To get the normal distances a_i between the direction of the normal \vec{n}_c and the center of mass \vec{X}_i for eccentric bodies in order to trigger a rotational motion, the distance vector between the point of collision \vec{C} and the center of mass \vec{X}_i is projected onto the tangential part of the local coordinate system. All quantities can be seen in Figure 2.2.

In the following, the momentum balance for the collision is built such that the momentum to be exchanged p_{ij} is yield out of

$$p_{ij} = (1 + e) \frac{U_{in} + a_i \omega_i - (U_{jn} + a_j \omega_j)}{\frac{1}{M_i} + \frac{a_i^2}{I_i} + \frac{1}{M_j} + \frac{a_j^2}{I_j}}, \quad (2.11)$$

see also Gross et al. (2008).

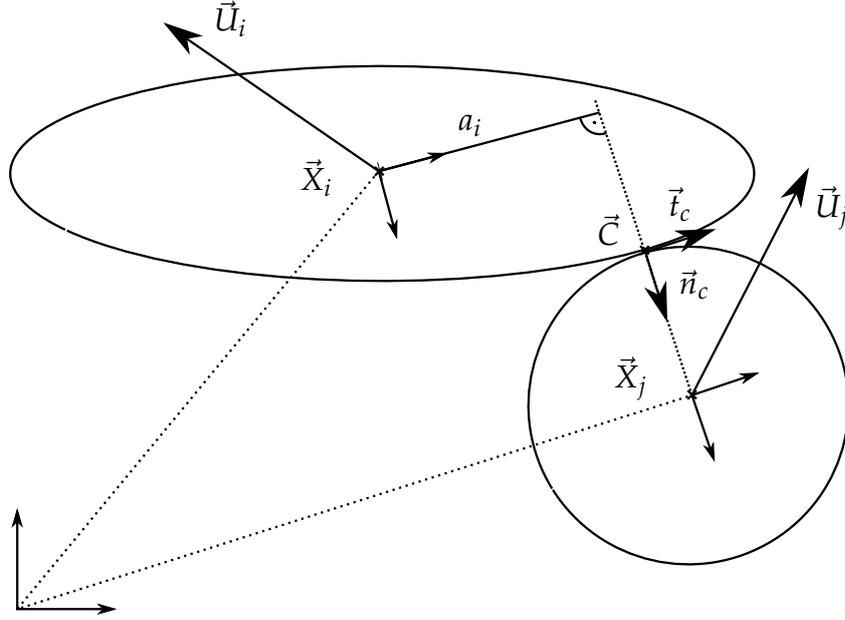


Figure 2.2: Eccentric collision between an ellipse i and a circle j .

Here, e is the coefficient of restitution, which is between $e = 0$ if the collision is plastic and $e = 1$ if the collision is fully elastic.

With p_{ij} , the new normal and tangential velocities \bar{U}_{in} , \bar{U}_{it} and \bar{U}_{jn} , \bar{U}_{jt} after collision can be determined using (2.12).

$$\bar{U}_{in} = U_{in} - \frac{p_{ij}}{M_i}, \quad \bar{U}_{it} = U_{it} \quad (2.12a)$$

$$\bar{U}_{jn} = U_{jn} + \frac{p_{ij}}{M_j}, \quad \bar{U}_{jt} = U_{jt} \quad (2.12b)$$

Note that because the surface of the particles is assumed to be smooth, the tangential part of the velocities will not be changed. Furthermore, the rotational velocity of both particles is determined using distances a_i and a_j together with p_{ij} , as it can be seen in

$$\bar{\omega}_i = \omega_i + a_i \frac{p_{ij}}{I_i}, \quad (2.13a)$$

$$\bar{\omega}_j = \omega_j - a_j \frac{p_{ij}}{I_j}. \quad (2.13b)$$

In the end, the normal and tangential parts of the velocities after collision have to be re-projected onto the global coordinate system:

$$\vec{U}_i = U_{in} \cdot \vec{n}_c + U_{it} \cdot \vec{t}_c \quad (2.14a)$$

$$\vec{U}_j = U_{jn} \cdot \vec{n}_c + U_{jt} \cdot \vec{t}_c \quad (2.14b)$$

Above, only particle-particle collisions are considered. For particle-wall collisions in (2.11), all quantities of the second particle are set to zero. The remaining part of the equation is kept as it is.

To connect this with the particle movement, collision forces will be formally treated like an additional force in (2.7). Moreover, it is important to mention, that the particle-wall collision cannot be conservative in terms of momentum because the wall is treated like a particle with infinite mass. Therefore, the model only conserves kinetic energy during particle-wall collisions.

To conclude, the two subproblems of fluid flow in Section 2.1 and particle movement in Section 2.2 are coupled together using the coupling technique described in Section 2.3 to yield a complete mathematical model.

3 State of the art

A literature review for general particle simulation methods will be given, focusing on the IBM approach. Then, the splitting approach will be introduced shortly. Afterwards, a short review on collision models and possible collision detection approaches will be given, which also includes some techniques from computer graphics. Furthermore, selected fictitious domain methods from the literature will be explained in detail to point out the possibilities of coupling fluid and particle solver depending on using either volume or surface forcing.

3.1 Literature review¹

Methods for particulate flows can be separated in two general groups: The first one is the so called Lagrangian approach, which uses a mesh fitted to the particle surfaces. As the mesh can move arbitrary in the fluid, those methods are called arbitrary Lagrangian Eulerian (ALE). The ALE method was used for particulate flows by Hu et al. (1991); Hu (1996) and Maury (1996, 1999). The second group are immersed boundary methods. The first IBM was proposed by Peskin (1972) in the field of FSI for simulating flow patterns around heart valves. The new feature of this method was, that all calculations were done on a fixed cartesian grid. It was not needed to remesh and project the solution onto the new grid in every timestep in order to be conform with the geometry. The success of his method was to impose the influence of the immersed boundary on the flow without remeshing. In the following years, various modifications on this method have been proposed and an overview can be taken from Mittal and Iaccarino (2005). Immersed boundary methods are commonly used for FSI problems including particulate flows. A broad review on IBM for FSI can be found in the work of Hou et al. (2012).

In context of particulate flows, IBM can be further distinguished by the coupling between fluid and particle interactions. The two variants are implicit and explicit coupling schemes. In the implicit coupling approach the forcing is incorporated into the flow equations before discretization. Important work in this field was done by Glowinski et al. (1999, 2001) using a fictitious domain method with distributed Lagrange multipliers and Patankar et al. (2000) using a stress distributed Lagrange multiplier ansatz to model the coupling forces. In the second approach the forcing is introduced after discretization.

First, the IBM was extended to Stokes flow around suspended particles by Fogelson and Peskin (1988) and Navier-Stokes flow around fixed cylinders by Lai and Peskin (2000). The number of particles was increased by the scheme proposed by Höfler and

¹Modified and extended version of (Krause and Kummer (2017), Section 1)

Schwarzer (2000). Further work has been done by Wan and Turek (2006) who proposed a multigrid fictitious domain method. To track the particles a volume based integral function is used, which was first proposed by Duchanoy and Jongen (2003). In this context, an alternative scheme was proposed by Uhlmann (2005), who combines Peskin (2002) regularized delta function approach with direct forcing in a finite-difference context. The IBM was also used in a Lattice-Boltzmann context, e.g. by Feng and Michaelides (2004).

An important issue which should be mentioned in terms of immersed boundary methods with moving domains is the occurrence of spurious pressure oscillations. These oscillations are present in many different IBM (Luo et al., 2009, 2010; Liao et al., 2010; Uhlmann, 2005; Mittal et al., 2008). Seo and Mittal (2011) found out that the reason for those oscillations lie in a violation of the conservation law due to the appearance and disappearance of cells at the interface. They suggest to apply a cut cell approach with virtual cell merging to enforce mass conservation. Further attempts to eliminate pressure oscillations have been made by Luo et al. (2009, 2010); Liao et al. (2010).

For decades, FVM have dominated the CFD community not only for single phase problems but for multi-phase problems like water-air interaction and particulate flows. In contrast, DG, first proposed by Reed and Hill (1973), became popular because of their ability to use higher-order ansatz spaces, like FEM, but still preserve conservation properties by definition, like FVM. DG also have several additional advantages, e.g. it is easy to handle hanging nodes and local refinements because of their discontinuous ansatz spaces at cell boundaries.

The work in this thesis is based on the extended discontinuous Galerkin (XDG) approach by Kummer (2016). Here, a sharp-interface representation is used. In order to treat the problem of high condition numbers for arbitrary small cut cells a cell-agglomeration procedure is employed. Using such a sharp-interface representation shifts the problem of accuracy and efficiency to the quadrature on those cut cells. For this, we use the hierarchical moment-fitting (HMF) first proposed by Müller et al. (2013) and later extended in the work of Kummer (2016). To the best of the authors knowledge, there is no work using cut cell/extended DG methods with HMF in connection with immersed boundaries to tackle particulate flow problems. However, extensive work in case of extended discretization methods has been done in context of extended finite element method (XFEM), first introduced by Moës et al. (1999) and later used for fluid dynamics by Groß and Reusken (2007). Beside other authors working in the field of XFEM, the first actual cut cell DG method was presented by Bastian and Engwer (2009).

3.2 Splitting scheme

For efficient time discretization methods like the one presented in this thesis, are based on the so called splitting scheme. The splitting essentially coupled fluid and particle problem together and contains the following steps:

1. The position of the particle $\vec{X}_j(t)$ within the fluid is given at time t^n . Additionally the coupling conditions at the boundaries of the fluid domain Ω_f as well as those on the particle surface Γ_j are given at t^n . The Navier-Stokes equations (2.5) are solved by using a common method for partial differential equations like FVM, FEM or DG to gain velocity and pressure at t^{n+1}
2. The hydrodynamic forces acting on the particle are calculated at t^{n+1}
3. The particles are moved with respect to the hydrodynamic forces calculated previously. The new boundary conditions on the particle surface are imposed and the scheme is repeated by setting $t^n := t^{n+1}$

Different approaches for the calculation of hydrodynamic forces in step 2 and how to imply boundary conditions, also in context with different spatial- and temporal discretizations have been made. The main details will be presented in Section 3.4.

3.3 Treatment of collisions

In the previous section, the splitting approach has been introduced. Now, it will be focused on a methods for treating particle-particle and particle-wall collisions. Specifically, this focuses in step 3 of the splitting, in which particles are moved respect to hydrodynamic forces. Thus, in case of a collision an additional force term applies.

Firstly, the modeling of collisions has to be distinguished from efficient collision detection in the solver. Thus, it will be focused on collision models based on the repulsive and lubrication forces between particles during collisions and models based on conservation of momentum. In the following, efficient collision detection in the technical field of particle simulation as well as some basics about collision detection in computer graphics will be presented.

3.3.1 Collision models

There are several approaches to treat collisions in particle simulations. However, there is the assumption that in finite-time particle-particle and particle-wall collisions in a continuous setting do not take place. This results out of repulsive and lubrications forces preventing collisions in terms of sufficiently viscous fluids (Glowinski et al., 2001; Glowinski, 2003). Nevertheless, collisions have to be modeled for numerical simulations to avoid decreasing the time step size, as well as the element size, if particles come too close. This would lead to a massive increase in computational costs as soon as particle collisions occur in the simulation.

For common collision models, one can distinguish repulsive force, lubrication force and models based on momentum conservation. The first two add a repulsive force to the right-hand side of the NEE (2.7). The latter is based on the equations of motion and a balance of total momentum at the point of contact. Here, the models which are described can be extended to particles with arbitrary shape quite easy. However, the extension shifts the computational effort to determining the distance between both particles efficiently, which is more expensive than for, e.g. spherical particles.

Repulsive force

The following collision model was developed by Glowinski et al. (2001) and is based on repulsive forces between colliding particles. It is assumed, that collisions have a smooth nature, meaning their particle velocities coincide at the point of contact. Additionally, it has to be ensured that the particles do not overlap each other at any time in the simulation. For circular particles the repulsive force \vec{F}_{rep} has to fulfill the following properties

$$\vec{F}_{rep} \text{ parallel to } \overrightarrow{X_i X_j}, \quad (3.1a)$$

$$\|\vec{F}_{rep}\| = 0 \text{ if } d_{ij} \geq R_i + R_j + \rho, \quad (3.1b)$$

$$\|\vec{F}_{rep}\| = \frac{c_{ij}}{\epsilon} \text{ if } d_{ij} = R_i + R_j. \quad (3.1c)$$

In (3.1), \vec{X}_i is the center of mass particle i , R_i is the radius and $d_{ij} = \|\overrightarrow{X_i X_j}\|$ is the distance between both center of masses. c_{ij} is a scaling factor and ϵ a small positive number depending on grid size. Additionally to the properties of (3.1), the model has to satisfy the behavior of Figure 3.1. This means a decreasing behavior between maximum and minimum value of repulsive force. In Figure 3.1, ρ is the range where the repulsive force model is acting and is chosen to be Δh , the minimum mesh size for the spatial discretization of the velocity in the original literature. Further information on how to choose this parameter can be found in Glowinski et al. (2001).

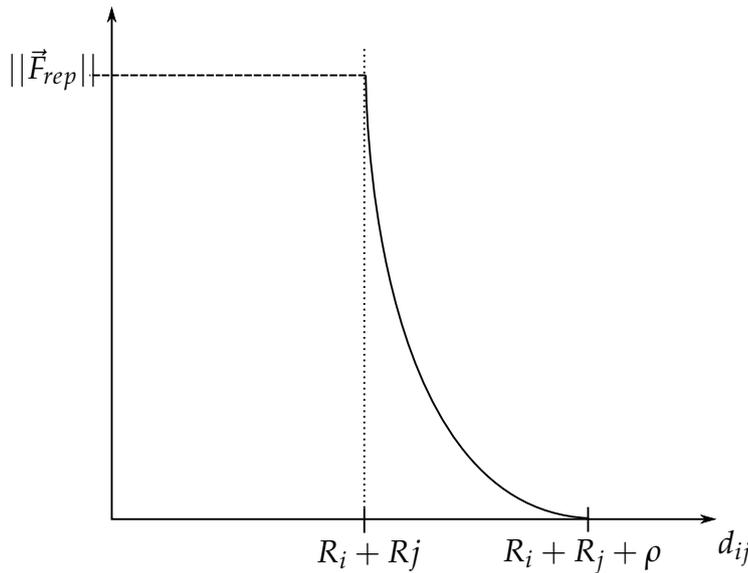


Figure 3.1: Force depending on distance in the model of Glowinski et al. (2001).

The aforementioned method does not allow the particles to overlap each other. However, overlapping can occur in numerical calculations, if the time step size is not adapted when particles come close to each other. As a result, Wan and Turek (2006) proposed a modified definition of the short-range repulsive force model where particles

can come arbitrary close and are even allowed to overlap. A schematic representation of the repulsive force with overlapping region can be seen in Figure 3.2.

$$\vec{F}_{\text{rep}} = 0 \text{ if } d_{ij} \geq R_i + R_j + \rho \quad (3.2a)$$

$$\vec{F}_{\text{rep}} = \frac{1}{\epsilon_p} (\vec{X}_i - \vec{X}_j) (R_i + R_j + \rho - d_{ij})^2 = \text{if } d_{ij} \geq R_i + R_j \quad (3.2b)$$

$$\vec{F}_{\text{rep}} = \frac{1}{\epsilon'_p} (\vec{X}_i - \vec{X}_j) (R_i + R_j - d_{ij}) \text{ if } d_{ij} \leq R_i + R_j \quad (3.2c)$$

In (3.2) ρ is again the range of the repulsive force and can be chosen to be a factor of 0.5 to 2.5 of the meshsize Δh . Further, ϵ_p and ϵ'_p are small stiffness parameters which are chosen to be $\epsilon_p \approx (\Delta h)^2$ and $\epsilon'_p \approx \Delta h$ if the ratio of both densities is around 1 and the fluid is sufficiently viscous.

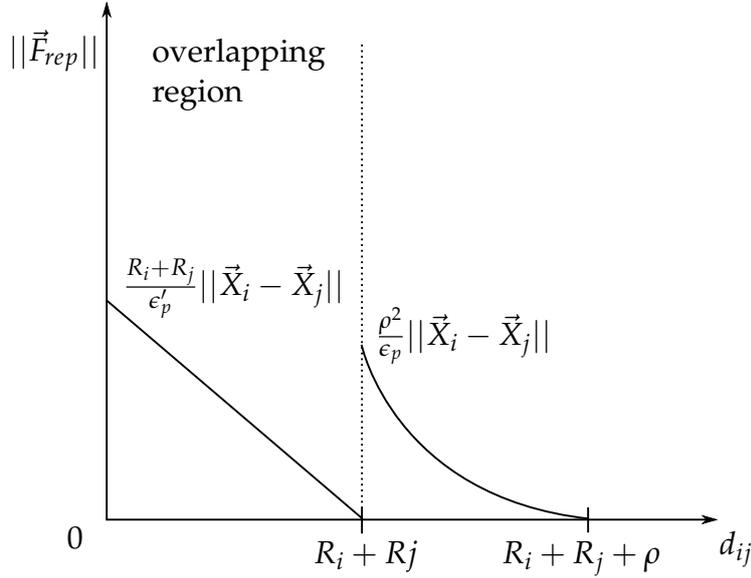


Figure 3.2: Force depending on distance in the model of Wan and Turek (2006), here overlap is possible.

The model can also be used for particle-wall collisions with some small modifications. In (3.3) $d'_i = |\vec{X}_i - \vec{X}'_i|$ is the distance between the center of mass and the center of an imaginary particle at the wall. The stiffness parameters for particle-wall collisions are chosen to be half of the particle-particle parameters, eg. $\epsilon_w = \epsilon_p/2$ and $\epsilon'_w = \epsilon'_p/2$.

$$\vec{F}_{\text{rep}} = 0 \text{ if } d'_{ij} > 2R_i + \rho \quad (3.3a)$$

$$\vec{F}_{\text{rep}} = \frac{1}{\epsilon_w} (\vec{X}_i - \vec{X}'_i) (2R_i + \rho - d'_i)^2 = \text{if } 2R_i \geq d'_i \geq R_i + R_j + \rho \quad (3.3b)$$

$$\vec{F}_{\text{rep}} = \frac{1}{\epsilon'_p} (\vec{X}_i - \vec{X}'_i) (2R_i - d'_i) \text{ if } d_i \leq 2R_i \quad (3.3c)$$

Lubrication force

If particles come very close to each other, a Poiseuille-type flow develops between them, leading to high local stress and pressure values. Here, the magnitude of the lubrication force strongly depends on the distance between the particles.

A very popular example of lubrication force models is the one of Maury (1997). Let \vec{C}_i and \vec{C}_j be the material points on the surface of two particles which are closest to each other. Further, $\dot{\vec{C}}_i$ and $\dot{\vec{C}}_j$ being the velocity at those points and \vec{n}_i and \vec{t}_i being the normal and tangential vector on the particle surface $\partial\Omega_i$, resp. $\partial\Omega_j$. All quantities are visualized in Figure 3.3. The lubrication force can be estimated by

$$\vec{F}_{\text{lub}} = [-\kappa_n(d_{ij})\vec{n}_c \otimes \vec{n}_c - \kappa_t(d_{ij})\vec{t}_c \otimes \vec{t}_c] \cdot (\dot{\vec{C}}_i - \dot{\vec{C}}_j), \quad (3.4)$$

where

$$\kappa_n(d) = \mu_n \frac{1}{d} \text{ and } \kappa_t(d) = \mu_t \ln\left(\frac{d_0}{d}\right). \quad (3.5)$$

In (3.5), μ_n and μ_t depend on geometrical aspects like curvature of the particle surface and on the viscosity of the fluid. Further, d_0 will be chosen according to the characteristic size of the particles. In his publication Maury (1997) assumes $\kappa_n(d)$ and $\kappa_t(d)$ to be given functions. It is also important that, like in the repulsive force models, a force will only act if d is greater than a given value d_0 .

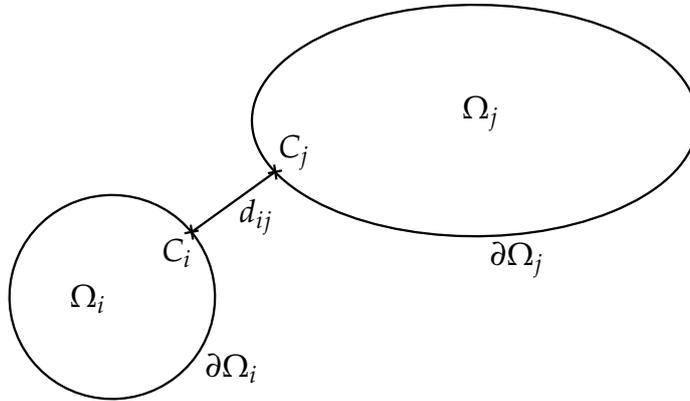


Figure 3.3: Particle quantities used by Maury (1997).

Conservation of momentum

Using the basics of rigid particle dynamics from Section 2.4, collision models based on the conservation of momentum can be constructed. Those models do not depend on an a-priori stiffness parameter. Ardekani and Rangel (2008) published a method based on the conservation of momentum for circular shaped particles with a Lagrange-Multiplier method. They consider only centric collisions, where both collision forces

act in the direction of the center of masses. This results in no torque acting on the rigid particle.

For this method, the velocities at the points of contact are split into normal and tangential parts of the particle surfaces $\vec{U} = U_n \vec{n}_c + U_t \vec{t}_t$. For tangential components of the velocities U_{t1} and U_{t2} momentum is conserved as well as the angular momentum for each particle separately. For both normal components the momentum balance renders down to

$$U_{n1}^+ = \frac{e(U_{n2}^- - U_{n1}^-)M_2}{M_1 + M_2} + \frac{M_1 U_{n1}^- + M_2 U_{n2}^-}{M_1 + M_2} \quad (3.6a)$$

$$U_{n2}^+ = \frac{e(U_{n2}^- - U_{n1}^-)M_1}{M_1 + M_2} + \frac{M_1 U_{n1}^- + M_2 U_{n2}^-}{M_1 + M_2} \quad (3.6b)$$

Here, the superscripts $-/+$ denote quantities right before and after the collision and M is the mass of the rigid particle. e is the coefficient of restitution.

For the numerical treatment of collisions Ardekani and Rangel (2008) simply calculate the contact force between both particles based on the rigidity force. If its magnitude is negative the collision process is ended since particles can not apply a tensile force to each other. The process is triggered if the distance between both particles is equal to Δr_{min} . Here, Δr_{min} denotes the roughness height of the particle. If no collision occurs and the particles are not in contact, the sum of all forces over each particle, excluding gravitational and hydrodynamical forces, renders to be 0.

3.3.2 Efficient collision detection

Efficient and accurate collision detection becomes important if the number of particles increases significantly and additionally the shape of the particle is arbitrary. In a brute force method, every particle is compared with every other particle in order to determine the distance between those two. As a result of a potential collision, the collision procedure can be triggered. This might be done at low computational costs even for a few hundred of spherical particles. However, for particles with arbitrary shape each interface point of one particle has to be compared to each interface point of another particle. This results in huge computational load and is not feasible even for few particles.

In the following, some basic methods for efficient collision detection will be presented. This section mainly contains basic algorithms used in computer graphics and computational geometry. In the end, an algorithm of Sigurgeirsson et al. (2001) will be presented, which is related to similar problems compared to this thesis.

Various methods have been developed for collision detection. Some of them mainly tailored to specific geometrical objects or to applications like physical based simulations (multi-body dynamics, particulate flows) and computer graphics (e.g. animation for video games). A good overview can be found in Jiménez et al. (2001), which is the base

of this short survey. In the following, the basics of collision detection algorithms for four general tasks will be mentioned:

i) How to detect collisions in a discrete time setting?

First, it is important to determine when a collision between a given set of moving rigid bodies occurs in the next time steps. In the method of spatio-temporal intersection a 4D space-time volume is being constructed along to its trajectory (Jimenez and Torras, 1995). If two volumes intersect each other, a collision has been detected. This method works for various geometries, however it is computational expensive because of the construction of complex space-time volumes. Thus other methods have been developed.

Avoiding the complex volume computation the volume is transferred into a lower-dimensional subspace by the swept volume approach. The swept volume contains all points occupied by a moving object during a time period. Nevertheless, it is possible that those constructed volumes intersect but a collision will not take place. As a result the relative velocity between those bodies is considered. Meaning one volume is kept fixed and the other one is moved with its relative velocity. However, also computing the swept volumes is expensive such that usually first only convex objects are considered (Foisy and Hayward, 1994). As soon as those global volumes intersect, the trajectories of all parts are considered. A simplified approach for simple shapes and trajectories has been published by Herman (1986).

Further, there is the method of multiple interference detection. Here, the trajectories of particles are sampled and tests for interference at particular sampling points are made. The choice of the sampling interval is crucial, because a collision can be missed. Ideally, the sample time can be calculated out of the distance between those bodies and their relative velocities. Most simple approaches have been published by Cameron (1985) and Culley and Kempf (1986).

Moreover, the time of collision can be tracked analytically if the trajectories are functions of time. With this method, general polyhedral shaped can be checked for with increasing polynomial degree of time. Rotational motion can also be taken into account because the resulting function in this case additionally contains trigonometric functions. For triangulated surfaces the work of Moore and Wilhelms (1988) can be mentioned.

ii) How to find intersections between bodies?

To find intersections it has to be distinguished between convex and non-convex polyhedral. It has to be mentioned that strategies for convex bodies have a significantly better performance than those for general bodies (Bouma and Vaněček, 1991). One popular approach is to check if an edge of one surface is piercing another face which becomes simple, if bodies have only convex faces. For general shapes there exist different strategies: First, it is often possible to split non-convex shapes into smaller convex ones and apply the strategy mentioned above. Further, it is known that for non-convex

faces a simple two-step test can show if an edge intersects a face, see Gilbert and Foo (1989).

Even if the basic interference tests are computationally cheap, it is of high gain to use information about object geometry and movement direction to delimit the number of interference tests to only parts of the objects. Various methods have been published for distance calculation between convex, non-convex and parametric surfaces which only consider parts of the objects. These methods can be split in two general classes, geometrical and optimization approaches. The first determine the closest points of two objects by calculation the euclidean distance between them, e.g. see Dobkin and Kirkpatrick (1990) and Cameron and Culley (1986). In the second approach distance is viewed as a function to be minimized (Bobrow, 1989).

For reducing the number of tests a strategy is very popular in computer graphics: The use of bounding volumes. In this method, bodies will be represented by a number of bounding volumes which have simple geometrical shape. This approach can also be used hierarchical, such that the area of a possible collision can be split into more bounding volumes again. This leads most of the time to a good prediction of the collision area at the first levels of hierarchy and closest distance computations can be restricted to that area. Those methods differ in the hierarchical structures of the underlying volumes. For example Hamada and Hori (1996) use a octree structure whereas also tetrahedral meshes (Klosowski et al., 1998) and regular grids (Garcia-Alonso et al., 1994) are being used.

If two bodies come close to each other with translational and rotational motion there exist various possible collision possibilities. Here, the back-face culling technique has to be mentioned (Vanekckek, 1994). In this method, the current relative velocity vectors of the rigid bodies is projected onto the normal-vectors of the particle surfaces. A face is culled, if the projection is negative, see Figure 3.4. Jiménez et al. (2001) state, that in average half of the faces of two polyhedra can be neglected with this method.

iii) The distances between which particles have to be computed?

For systems containing a huge number of rigid bodies it is also important to restrict the collision detection algorithm only to bodies which can collide in the next timestep because of their distance and relative motion. To tackle this issue, Tornero et al. (1991) published the concept of awareness. For every particle, they determine an increasing value of awareness containing position, relative velocity and acceleration. Basically, the lowest number of awareness denotes objects which are most likely to collide during the simulation. Additionally, they use buckets to cluster bodies with certain awareness levels such that bodies which are more likely to collide get also updated more often. Of course, they have been published many other techniques with large similarities, such that also a heap can be used to store those event queues like it will be prescribed in the particular method of Sigurgeirsson et al. (2001).

Since the methods presented are only a rough outline, more details can be found in Jiménez et al. (2001) or van den Bergen (2003).

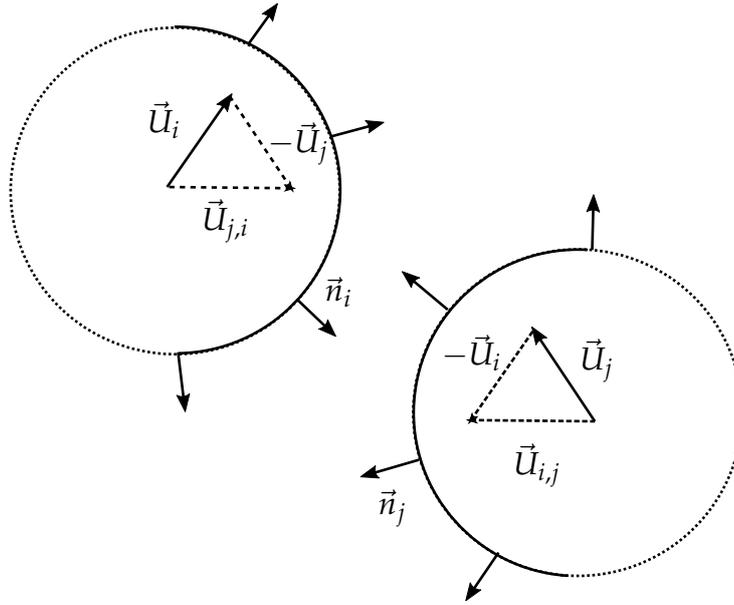


Figure 3.4: Back-face culling technique for two circular bodies with culled surfaces according to their relative velocities.

Method of Sigurgeirsson et al. (2001)

An algorithmic approach tailored to the application of fluid dynamics was proposed by Sigurgeirsson et al. (2001) and will be mentioned in this context, because its application is very close to the moving particle simulations in this thesis.

They use a coarse grid approximation of their domain and assign those coarse cells to so-called elements, which is very similar to a domain decomposition approach. Additionally, they handle a data structure in which the number of particles contained in a specific (coarse) element is stored. Now, with given velocities the NEE can be integrated in time to predict new particle positions. The big advantage is that a particle j has only to be compared to particles in the same element and surrounding elements in order to potentially trigger a collision model. Particles which do not share an element or neighboring element do not have to be considered. Subsequently, also a transfer of particles between elements without collisions has to be proceeded and the data structure has to be updated frequently. A smart choice of decomposing the total computational domain can therefore lead to large saving of computational costs.

As an improvement for coupled particle-flows, naming type (III) flows in the categorization of Sigurgeirsson et al. (2001), in the beginning they solve the equation describing the fluid behavior using old particle positions X^n and velocities U^n . Then they incorporate the fluid velocity u^{n+1} at the new time step t^{n+1} into the integration of the equation of motion and yield a more sophisticated prediction of the new particle positions X^{n+1} .

3.4 Immersed boundary methods

As mentioned in the beginning of this chapter, the following part will point out a few approaches for the coupling of fluid and particle equation solve strategies in more detail. This is particular useful to give context to the cut cell IBM which will be presented afterwards. The review will be split in two parts, methods with volume and surface coupling. All methods mentioned in this section apply a smooth interface approach in which the boundary between fluid and particles is not explicitly known and no surface integration is evaluated.

3.4.1 Coupling forces at volume

In the following methods, the particle equations are implicitly coupled with the NSE using Body-force distributed and stress distributed Lagrange multiplier techniques. The goal is to solve one coupled system of equations for physical properties of fluid and particle.

The first work to mention in connection with implicit coupling approaches in terms of moving particle flows is the one of Glowinski et al. (2001). In this approach the Navier-Stokes (2.5) is coupled with the Newton-Euler (2.7) which describes the particle motion. The constraint of particle motion in this method is described by (2.9).

They assume a two-phase flow with a fluid phase and a particle phase. Therefore each particle Ω_j is made of a homogeneous material of density ρ_j which differs from the fluid density ρ_f . When taking into account that any particle velocity field \vec{v} verifies $\nabla \cdot \vec{v} = 0$ and $D(\vec{v}) := \frac{1}{2}(\nabla \vec{v} + \nabla \vec{v}^T) = 0$ the following formulation is yield:

For a.e. $t > 0$, find $\vec{u}(t), p(t), \{\vec{V}_j(t), \vec{X}_j(t), \vec{\omega}_j(t)\}_{j=1}^J$, such that

$$\begin{aligned} & \rho_f \int_{\Omega} \left[\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla) \vec{u} \right] \cdot \vec{v} \, dx - \int_{\Omega} p \nabla \cdot \vec{v} \, dx + \nu \int_{\Omega} D(\vec{u}) : D(\vec{v}) \, dx \\ & + \sum_{j=1}^J \left(1 - \frac{\rho_f}{\rho_j} \right) \left[M_j \frac{d\vec{U}_j}{dt} \cdot \vec{Y}_j + (\mathbf{I}_j \frac{d\vec{\omega}_j}{dt} + \vec{\omega}_j \times \mathbf{I}_j \vec{\omega}_j) \cdot \vec{\theta}_j \right] \\ = & \rho_f \int_{\Omega} \vec{g} \cdot \vec{v} \, dx + \sum_{j=1}^J \left(1 - \frac{\rho_f}{\rho_j} \right) M_j \vec{g} \cdot \vec{Y}_j, \quad \forall \{\vec{v}, \vec{Y}, \vec{\theta}\} \in \tilde{W}_0(t) \text{ in } \Omega \end{aligned} \quad (3.7a)$$

$$\int_{\Omega} q \nabla \cdot \vec{u} \, dx = 0, \quad \forall q \in L^2(\Omega), \quad (3.7b)$$

$$\vec{u} = \vec{g}_0 \text{ on } \Gamma, \quad (3.7c)$$

$$\vec{u}(\vec{x}, t) = \vec{V}_j(t) + \omega_j(t) \times (\vec{X}_j(t) - \vec{x}), \quad \forall \vec{x} \in \Omega_j(t), \forall j = 1, \dots, J. \quad (3.7d)$$

$$\frac{d\vec{X}_j}{dt} = \vec{V}_j, \quad (3.7e)$$

$$\Omega_j(0) = \Omega_{0j}, \quad \vec{V}_j(0) = V_{0j}, \quad \omega_j(0) = \omega_{0j}, \quad \vec{X}_j(0) = X_{0j}, \quad \forall j = 1, \dots, J. \quad (3.7f)$$

$$\vec{u}(\vec{x}, 0) = \vec{u}_0(\vec{x}), \quad \forall \vec{x} \in \Omega \setminus \bigcup_{j=1}^J \bar{\Omega}_{0j} \text{ and } \vec{u}(\vec{x}, 0) = \vec{V}_{0j} + \omega_{0j} \times (\vec{X}_{0j} - \vec{x}), \quad \forall \vec{x} \in \bar{\Omega}_{0j}. \quad (3.7g)$$

with space $\tilde{W}_0(t)$ defined by

$$\begin{aligned} \tilde{W}_0(t) &= \{ \{v, Y, \theta\} \mid \vec{v} \in (H_0^1(\Omega))^d, \\ Y &= \{\vec{Y}_j\}_{j=1}^J, \quad \theta = \{\vec{\theta}_j\}_{j=1}^J, \text{ with } \vec{Y}_j \in \mathbb{R}^D, \vec{\theta}_j \in \mathbb{R}^3, \\ \vec{v}(\vec{x}, t) &= \vec{Y}_j + \vec{\theta}_j \times (\vec{X}_j - \vec{x}) \quad \text{in } \Omega_j(t), \quad \forall j = 1, \dots, J \}. \end{aligned} \quad (3.8)$$

For \vec{u} and p it can be assumed that $\vec{u}(t) \in (H^1(\Omega))^D$ and $p(t) \in L^2(\Omega)$. In order to relax the particle constrained (3.7d), Glowinski et al. (2001) employ a family $\vec{\lambda}_{j=1}^J$ of Lagrange multipliers so that $\vec{\lambda}_j(t) = \Lambda_j(t)$ with

$$\vec{\lambda}_j(t) \in (H^1(\Omega_j(t)))^d, \quad \forall j = 1, \dots, J. \quad (3.9)$$

By the help of Lagrange multipliers the movement of the particle phase is restricted to rigid body motion:

For a.e. $t > 0$, find $\vec{u}(t), p(t), \{\vec{V}_j(t), \vec{X}_j(t), \vec{\omega}_j(t), \vec{\lambda}_j(t)\}_{j=1}^J$, such that

$$\vec{u}(t) \in (H^1(\Omega))^D, \vec{u}(t) = \vec{g}_0(t) \text{ on } \Gamma, p(t) \in L^2(\Omega), \quad (3.10a)$$

$$\vec{V}_j(t) \in \mathbb{R}^D, \vec{X}_j(t) \in \mathbb{R}^D, \vec{\omega}_j(t) \in \mathbb{R}^3, \vec{\lambda}_j(t) \in \Lambda_j(t), \quad \forall j = 1, \dots, J, \quad (3.10b)$$

and

$$\begin{aligned} &\rho_f \int_{\Omega} \left[\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla) \vec{u} \right] \cdot \vec{v} \, dx - \int_{\Omega} p \nabla \cdot \vec{v} \, dx \\ &+ \nu \int_{\Omega} D(\vec{u}) : D(\vec{v}) \, dx - \sum_{j=1}^J \langle \vec{\lambda}_j, \vec{v} - \vec{Y}_j - \vec{\theta}_j \times (\vec{X}_j - \cdot) \rangle_j \\ &+ \sum_{j=1}^J \left(1 - \frac{\rho_f}{\rho_j} \right) \left[M_j \frac{d\vec{U}_j}{dt} \cdot \vec{Y}_j + \left(\mathbf{I}_j \frac{d\vec{\omega}_j}{dt} + \vec{\omega}_j \times \mathbf{I}_j \vec{\omega}_j \right) \cdot \vec{\theta}_j \right] \\ &= \rho_f \int_{\Omega} \vec{g} \cdot \vec{v} \, dx + \sum_{j=1}^J \left(1 - \frac{\rho_f}{\rho_j} \right) M_j \vec{g} \cdot \vec{Y}_j, \quad (3.10c) \\ &\forall \vec{v} \in (H_0^1(\Omega))^D, \forall \vec{Y}_j \in \mathbb{R}^D, \forall \vec{\theta}_j \in \mathbb{R}^3, \end{aligned}$$

$$\int_{\Omega} q \nabla \cdot \vec{u} \, dx = 0, \quad \forall q \in L^2(\Omega), \quad (3.10d)$$

$$\langle \vec{\mu}_j, \vec{u}(t) - \vec{V}_j(t) - \omega_j(t) \times (\vec{X}_j - \cdot) \rangle_j = 0, \quad \mu_j \in \Lambda_j(t), \forall j = 1, \dots, J, \quad (3.10e)$$

$$\frac{d\vec{X}_j}{dt} = \vec{V}_j, \quad (3.10f)$$

$$\Omega_j(0) = \Omega_{0j}, \quad \vec{V}_j(0) = \vec{V}_{0j}, \quad \omega_j(0) = \omega_{0j}, \quad \vec{X}_j(0) = \vec{X}_{0j}, \quad \forall j = 1, \dots, J. \quad (3.10g)$$

$$\vec{u}(\vec{x}, 0) = \vec{u}_0(\vec{x}), \quad \forall \vec{x} \in \Omega \setminus \bigcup_{j=1}^J \bar{\Omega}_{0j} \text{ and } \vec{u}(\vec{x}, 0) = \vec{V}_{0j} + \omega_{0j} \times (\vec{X}_{0j} - \vec{x}), \quad \forall \vec{x} \in \bar{\Omega}_{0j}. \quad (3.10h)$$

Different choices for $\langle \vec{\mu}, \vec{v} \rangle_j$ relaxing the rigid body motion constraint can be made. Two of the most natural ones are the form

$$\langle \vec{\mu}, \vec{v} \rangle_j = \int_{\Omega_j} (\vec{\mu} \cdot \vec{v} + \delta_j^2 \nabla \vec{\mu} \nabla \vec{v}) \, dx, \quad \forall \vec{\mu} \text{ and } \vec{v} \in \Lambda_j(t) \quad (3.11)$$

and

$$\langle \vec{\mu}, \vec{v} \rangle_j = \int_{\Omega_j} (\vec{\mu} \cdot \vec{v} + \delta_j^2 [\nabla \vec{\mu} + (\nabla \vec{\mu})^T] : [\nabla \vec{v} + (\nabla \vec{v})^T]) \, dx, \quad \forall \vec{\mu} \text{ and } \vec{v} \in \Lambda_j(t), \quad (3.12)$$

where δ_j is a characteristic length (for example the particle diameter).

Another method for the interaction between particle and fluid was developed by Höfler and Schwarzer (2000). They use a penalty method instead of the multiplier technique described above. Thus, in both methods the particle motion is represented by means of (2.9).

A different approach for the implicit coupling was developed by Patankar et al. (2000). In this method, no interface force terms in the equation system appear. Instead an additionally stress is used to enforce the rigid particle motion just like pressure in a fluid. Patankar et al. (2000) uses an alternative fictitious domain approach, in which the rigidity of the particles is incorporated by the constraint $D(\vec{u}) = 0$. The corresponding variational formulation then reads:

For $t > 0$, find $\vec{u} \in W_{uD}$, $p \in L_0^2(\Omega)$, $\vec{\lambda} \in H^1(\Omega_j(t))^d$ satisfying

$$\begin{aligned} & \rho_f \int_{\Omega} \left[\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla) \vec{u} - \vec{f} \right] \cdot \vec{v} \, dx - \int_{\Omega} p \nabla \cdot \vec{v} \, dx + \int_{\Omega} q (\nabla \cdot \vec{u}) \, dx + \int_{\Omega} \boldsymbol{\tau} : D[\vec{v}] \, dx \\ & + \int_{\Omega_j(t)} (\rho_j - \rho_f) \left[\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla) \vec{u} - \vec{f} \right] \cdot \vec{v} \, dx + \int_{\Omega_j(t)} D[\vec{\lambda}] : D[\vec{v}] \, dx \\ & + \int_{\Omega_j(t)} D[\vec{\mu}] : D[\vec{u}] \, dx = 0 \end{aligned} \quad (3.13a)$$

$$\forall \vec{v} \in W_{0D}, \quad \mu \in H^1(\Omega_j(t))^D \text{ and } q \in L^2(\Omega_j(t))$$

where

$$W_{uD}(t) = \{ \vec{v} \in H^1(\Omega)^D : \vec{v} = \vec{u}_{\Gamma}(t) \text{ on } \Gamma_D \},$$

$$W_{0D} = \{ \vec{v} \in H^1(\Omega)^D : \vec{v} = 0 \text{ on } \Gamma_D \},$$

$$L_0^2(\Omega) = \{ q \in L^2(\Omega) \mid \int_{\Omega} q \, dx = 0 \}.$$

Again, $\vec{\lambda}$ is a Lagrange multiplier for the constraint $D(\vec{u}) = 0$, mentioned above. τ represents the extra stress tensor based on the deformation of the fluid at a given position. Thus, $\tau = 0$ inside Ω_j .

For the detailed derivation of the weak form the reader is referred to Patankar et al. (2000). The main point to state here is that in comparison with (3.7a) there are no explicit interface force terms or particle velocities present in (3.13a). This is, as described above, convenient for the simulation of particulate flows in three dimensions where the angular momentum equations become more complex.

3.4.2 Coupling forces at surface

Instead of applying the coupling forces at the volume, there exist method which apply forces at the particle surfaces. The surface approach is also close to the approach presented in this thesis and therefore some key literature will be presented in the following. An important contribution to the field of coupled methods was submitted by Duchanoy and Jongen (2003). The equation system consists of the NSE (2.5) and the constraint on the fluid velocity (2.9) which is imposed in the particle domain $\Omega_j(t)$. Those equations are discretized by using a standard FVM.

Nevertheless, an auxiliary function α_{pqr} is introduced in the following to distinguish between control volumes in the fluid and in the particle domain:

$$\alpha_{pqr} = \sum_{j=1}^N \alpha_{pqr}^j, \quad \text{with } \alpha_{pqr}^j = \begin{cases} 0, & \vec{x}_{pqr} \in \Omega_f(t) \\ 1, & \vec{x}_{pqr} \in \Omega_j(t), \end{cases} \quad (3.14)$$

where N is the number of grid nodes and \vec{x}_{pqr} denotes the center of the control volume (p, q, r) for all control volumes within the grid. The auxiliary function is needed to represent a smooth particle interface within a Cartesian grid. Also, by using α_{ijk} the force \vec{F} and torque \vec{T} acting on each particle can be calculated using a summation with

$$\vec{F}_j = \sum_{pqr \in K_h} \sigma_{pqr} \vec{n}_{pqr}^j \quad (3.15)$$

$$\text{and} \quad (3.16)$$

$$\vec{T}_j = \sum_{pqr \in K_h} (\vec{x}_{pqr} - \vec{X}_j) \times \sigma_{pqr} \vec{n}_{pqr}^j. \quad (3.17)$$

Here \vec{X}_j denotes the center of the particle and K_h is the computational grid. Important for the calculation of the hydrodynamic forces is the property of \vec{n}_{pqr}^j which is only non-zero at the interface between particle. Further it is important to state the the value of the gradient of α_{pqr} is proportional to the interfacial surface area in cell (p, q, r) .

$$\vec{n}_{pqr}^j = \alpha_{pqr}^j (\nabla \alpha_{pqr}^j). \quad (3.18)$$

The multiplication of α_{pqr} is needed in order to filter the contribution of cells lying at the fluid part of the interface.

In consequence, this approach is used to transfer the surface integral for evaluating the hydrodynamic forces to a volume integral over the whole domain. In Figure 3.5, which is extracted from the reference cited above, the representation of a particle in the FVM discretization can be seen. The algorithm of Duchanoy and Jongen (2003)

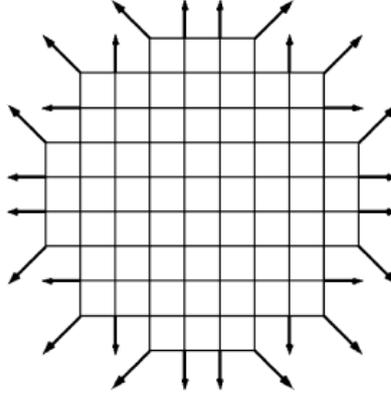


Figure 3.5: Shape of a particle (cells with $\alpha_{pqr} = 1$) with its normal vectors taken from Duchanoy and Jongen (2003).

uses a standard pressure correction scheme for computation. The particle position is updated with the intermediate velocity and the pressure field. More details of this specific method can be found in the literature cited above. Wan and Turek (2006) proposed a related approach in context of FEM. In their scheme, the calculation of hydrodynamic forces (3.19) differs slightly from the one of Duchanoy and Jongen (2003) but the same definition of the auxiliary function $\alpha_i(\vec{x})$ in (3.14) is being used. Instead of just summing up the values at the nodes along the boundary in a FVM, they use a standard Gaussian quadrature of corresponding high-order to solve a volume-integral, where $\nabla\alpha_i$ is only different from zero at the particle surface. Thus, the volume integral has only to be computed in one layer of mesh cells around the particle boundary. The conversion from surface integral to volume integral is an important feature, because the information about the particle boundary only lies implicitly in the grid and its reconstruction would be time consuming.

$$\vec{F}_j = - \int_{\Omega} \boldsymbol{\sigma} \cdot \nabla \alpha_i \, d\Omega \quad (3.19a)$$

$$\vec{T}_j = - \int_{\Omega} (\vec{X} - \vec{X}_j) \times (\boldsymbol{\sigma} \cdot \nabla \alpha_i) \, d\Omega \quad (3.19b)$$

For solving the equations, a discrete projection scheme which splits the coupled problem is used. Then, the non-linear equations in \vec{u} are treated by a non-linear iteration or a linearization technique. The pressure equations are solved by using an optimal multigrid solver.

The method of Uhlmann (2005) was published in a second-order finite-difference context and basically uses the delta function approach of Peskin (2002) with an additional direct formulation of the fluid-solid interaction force. The fluid and the particles are discretized differently. Fluid quantities are prescribed by Eulerian variables and particle

quantities are treated with Lagrangian variables. For each particle of number of total particles N_p embedded in the fluid region a number of N_L points is distributed evenly along the fluid-particle surface. Those points are called Lagrangian force points \vec{X}_l^j . They follow the particle motion and do not require additional tracking. Furthermore, a discrete volume ΔV_l^j is associated with each force point so that a thin shell around each particle is formed (see Figure 3.6). For circular particles and a given mesh size Δh the discrete volume is chosen to be

$$\Delta V_l^j = \frac{2\pi r_c \Delta h}{N_L}. \quad (3.20)$$

The shell makes it possible to apply a volume force to each particle. Moreover, the key

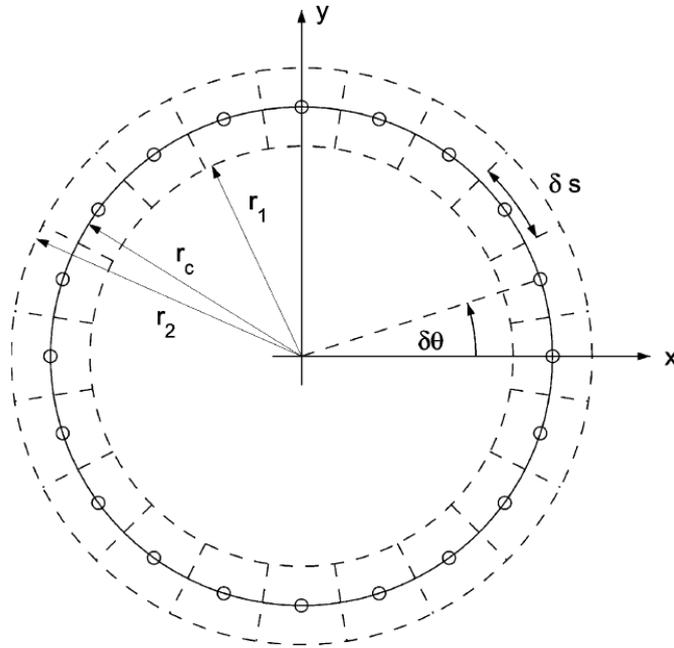


Figure 3.6: Shell with equidistant force points N_L and an associated volume ΔV_l^j taken from Uhlmann (2005).

feature of the method is the mapping between Lagrangian and Eulerian coordinates and vice versa using Peskin's delta function approximation. The mapping of velocity and force is done by

$$\vec{U}(\vec{X}_l^j) = \sum_{\vec{x} \in G_h} \vec{u}(\vec{x}) \delta_h(\vec{x} - \vec{X}_l^j) \Delta h^3 \quad \forall 1 \leq j \leq N_p; 1 \leq l \leq N_L \quad (3.21a)$$

and

$$f(\vec{x}) = \sum_{j=1}^{N_p} \sum_{l=1}^{N_L} F(\vec{X}_l^j) \delta_h(\vec{x} - \vec{X}_l^j) \Delta V_l^j \quad \forall \vec{x} \in K_h, \quad (3.21b)$$

where K_h is the grid, Δh is the mesh width and δ_h is a continuously differentiable function. Capital letters show Lagrangian quantities and the small ones Eulerian quantities.

The solver is based on a fractional-step method. A three-step Runge-Kutta scheme is used for the convective terms and the Crank-Nicolson method is used for viscosity.

The main advantage of using a combination of smooth transfer between Lagrangian and Eulerian coordinates and the direct forcing approach in the present method is, that it leads to less oscillatory particle forces than existing direct methods and has a higher efficiency compared to implicit methods.

Lastly, the descriptions in this chapter can only give a short overview of the different methods which have been published in context of IBM with particulate flows. For more information there has recently been published a review on this topic by Maxey (2017). It is important to state, that all methods described above use some sense of smooth interface representation. On the contrary, in the presently proposed method a sharp interface representation using cut cells is implemented.

4 Cut cell DG method for particulate flows

The main part of this chapter is the description of the presented method in this thesis, including discretization and collision modeling. Also some details of the method naming efficient quadrature and cell-agglomeration are presented. In the end, the current chapter will be shortly concluded.

4.1 Splitting scheme

At the beginning, the splitting technique to couple both, flow and particle problem will be introduced. Note that the numbering represents steps which have to be proceeded in each time step. The scheme is described in a two-dimensional setting for simplification reasons. Of course, the procedure is similar in three dimensions. The discretization of the flow problem will be presented in detail in Section 4.2.

1. Solve the NSE for \vec{u}^{n+1} and p^{n+1} with given initial and boundary conditions using a Picard or Newton linearization technique and a linear solver for the linearized equation system. Thus, the discretized NSE (2.5) in time is described by

$$\begin{aligned} \rho_f \left(\frac{3\vec{u}^{n+1}}{2\Delta t} - \frac{2\vec{u}^{n*}}{\Delta t} + \frac{\vec{u}^{n-1*}}{2\Delta t} + \vec{u}^{n+1} \cdot \nabla \vec{u}^{n+1} \right) \\ + \nabla p^{n+1} - \mu_f \Delta \vec{u}^{n+1} = \vec{f}^{n+1} \end{aligned} \quad (4.1a)$$

and

$$\nabla \cdot \vec{u}^{n+1} = 0 \quad \text{in } \Omega_f(t^{n+1}). \quad (4.1b)$$

2. Calculate the hydrodynamic force \vec{F}_j^{n+1} and torques \vec{T}_j^{n+1} acting on the j -th particle using the HMF on the particle surface Γ_j .
3. Potential particle-particle and particle-wall collision are triggered, after checking cell set intersections and distances d_{ij} . For details see Section 4.3.
4. Solve the NEE (2.7) for new particle velocities \vec{U}_j^{n+1} and rotational speed ω_j^{n+1} by using a second-order Crank-Nicolson method in time:

$$\vec{U}_j^{n+1} = \vec{U}_j^n + \left(\frac{\Delta M_j \vec{f}}{M_j} + \frac{\vec{F}_j^n + \vec{F}_j^{n+1}}{2M_j} \right) \Delta t, \quad (4.2)$$

$$\omega_j^{n+1} = \omega_j^n + \left(\frac{T_j^n + T_j^{n+1}}{2I_{jx}} \right) \Delta t, \quad (4.3)$$

where I_{jx} is the moment of inertia for rotation of a specific particle shape and ΔM_j is the mass difference between particle and fluid occupying the same volume.

5. Determine new particle positions \vec{X}_j^{n+1} and angles θ_j^{n+1} with

$$\vec{X}_j^{n+1} = \vec{X}_j^n + \vec{U}_j^n \Delta t + \frac{1}{2} \left(\frac{\Delta M_j \vec{f}}{M_j} + \frac{\vec{F}_j^n + \vec{F}_j^{n+1}}{2M_j} \right) \Delta t^2, \quad (4.4)$$

$$\theta_j^{n+1} = \theta_j^n + \omega_j^n \Delta t + \frac{1}{2} \left(\frac{T_j^n + T_j^{n+1}}{2I_{jx}} \right) \Delta t^2. \quad (4.5)$$

6. Move particle to the new position and extrapolate velocities at the new domain with

$$\vec{u}^{n*} = \vec{u}^n \text{ in } \Omega_f(t^{n+1}) \quad (4.6a)$$

and

$$\vec{u}^{n-1*} = \vec{u}^{n-1} \text{ in } \Omega_f(t^{n+1}). \quad (4.6b)$$

7. Apply velocity boundary conditions at the particle surfaces Γ_j following (2.9).

8. Proceed to the next time level by setting $t^n := t^{n+1}$ and restart with step 1.

In this context it should be noted, that using an explicit splitting method for the moving of boundaries within the grid renders the total scheme still to be first order in time only. Therefore the overall scheme renders to be of first order for moving domains and converges with $\mathcal{O}(\Delta t + h^{k+1})$. Thus Δt has to be decreased if the polynomial degree is increased.

Finally, in Figure 4.1 the solver procedure is presented in a flowchart. Of course, each step consist of further substeps, which will be presented in detail in this chapter. The overall convergence order in time is bound by the splitting approach in step 6.

4.2 Discontinuous Galerkin method for flow problem¹

In this section, the numerical background of the cut cell immersed boundary method within a discontinuous Galerkin context will be presented. It should be mentioned that the present solver is based on the work of Kummer (2016), who has applied DG two fluid phase flows and is implemented into the software package *BoSSS* (Kummer, 2012). First, some basic definitions and notations will be given. Afterwards, the discretization in space and time will be introduced. Then, a few words about the HMF method, first published by Müller et al. (2013), will be given. In extension to Krause and Kummer (2017), a collision model for the cut cell immersed boundary method will be presented after discretization.

¹Modified and extended version of (Krause and Kummer (2017), Section 3)

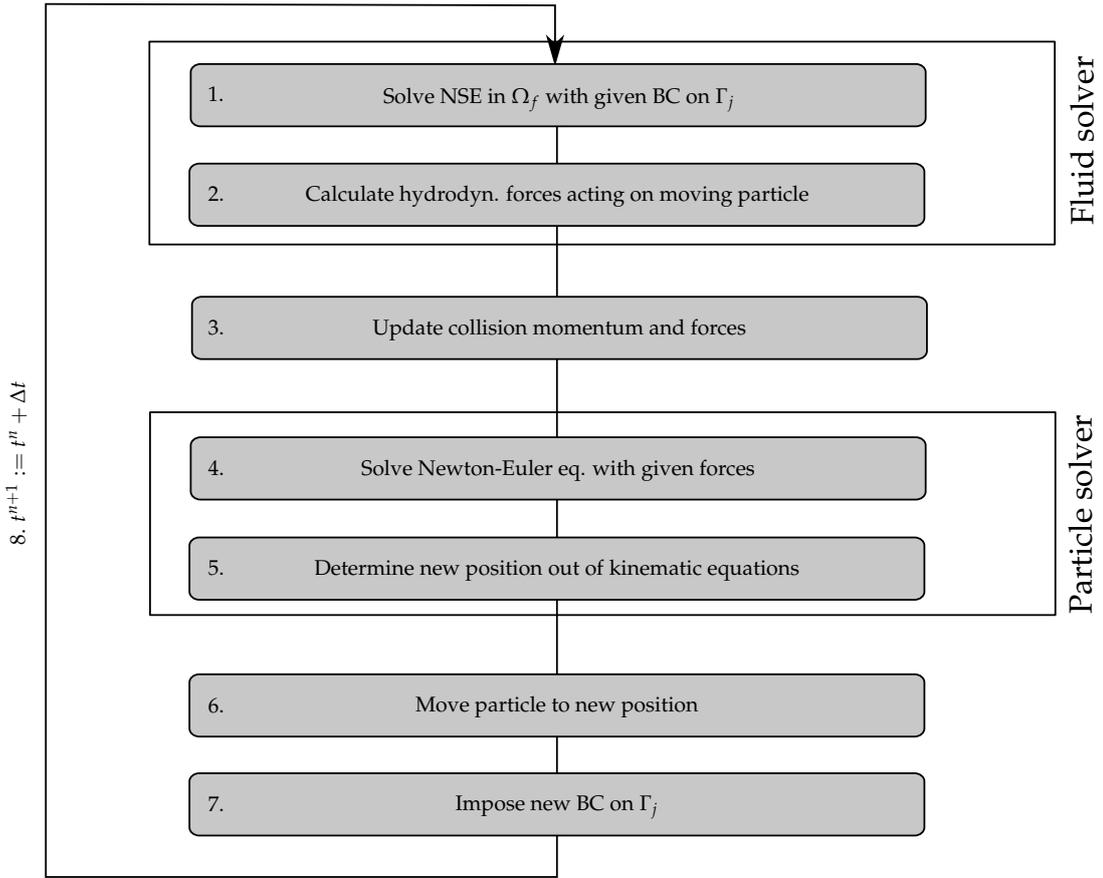


Figure 4.1: Solver process scheme using an explicitly coupled method between fluid and particle part.

Foremost, basic notations and definitions of the discontinuous Galerkin method have to be introduced:

- Let the computational domain $\Omega \subset \mathbb{R}^D$ be bounded Lipschitz.
- The numerical background grid $\mathcal{K}_h^B = \{K_1, \dots, K_n\}$. The cells $\{K_1, \dots, K_n\}$ should be of diameter $d \approx h$ and star-shaped with respect to a ball of radius $\rho \approx h$.
- The numerical grid $\mathcal{K}_h(t) = \mathcal{K}_h^B \cap \overline{\Omega}_f(t)$ containing fluid and cut cells with h being the maximum diameter of all cells. For the decoupled fluid subproblem $\Omega_f(t)$ results out of the method and is according to Chapter 2 assumed to be given.
- Let $\Gamma_I(t)$ be the union of all interior cell faces::

$$\Gamma_I(t) = \bigcup_j \partial K_j \cap \Omega_f(t).$$
- Let $\Gamma(t)$ be the union of all cell faces in the fluid region:

$$\Gamma(t) = \Gamma_I(t) \cup \partial \Omega_f(t). \text{ Here } \partial \Omega_f(t) = \Gamma_D \cup \Gamma_N \cup \bigcup_j \Gamma_j(t), \text{ see Section 2.1.}$$
- a normal field \vec{n}_Γ on $\Gamma(t)$ which defines the face orientation.

- Jump and average operators of a function ϕ on interior faces are denoted by:
 $[[\phi]] := \phi|_{K^+} - \phi|_{K^-}$ and $\{\phi\} := \frac{1}{2}\phi|_{K^+} + \frac{1}{2}\phi|_{K^-}$ on $\Gamma_I(t)$ and
for all boundary faces of $\Omega_f(t)$ by:
 $[[\phi]] := \phi|_{K^+}$ and $\{\phi\} := \phi|_{K^+}$ on $\Gamma \setminus \Gamma_I(t)$.
- The gradient ∇_h on the broken polynomial space: for $u \in C^1(\Omega_f)$, $\nabla_h u$ denotes the broken gradient on the domain Ω_f and $\nabla_h \cdot \vec{u}$ the broken divergence.

In order to formulate the discretization the broken polynomial space on cut cells (Kummer, 2016) is defined in the following way:

$$\mathbb{P}_k(\mathcal{K}_h(t)) := \{f \in L^2(\Omega_f(t)); \forall K \in \mathcal{K}_h(t) : f|_K \in \mathbb{P}_k(K) \text{ is polynomial, } \deg(f|_{K \cap \Omega_f(t)}) \leq k\}. \quad (4.7)$$

Note that because $\mathcal{K}_h(t)$ is time dependent for $\Omega_f(t)$, \mathbb{P}_k also depends on time. Furthermore a mapping to extrapolate the known velocities \vec{u}^α onto the new polynomial space is needed.

The extrapolation

$$\mathbb{E}\vec{u} : \Omega_f(t^\beta) \rightarrow \mathbb{R}^D \quad (4.8)$$

is defined element-by-element as follows:

- If $|K \cap \Omega_f(t^\beta)| = 0$, nothing has to be defined.
- If $|K \cap \Omega_f(t^\beta)| > 0$ and $|K \cap \Omega_f(t^\alpha)| > 0$, then $\mathbb{E}\vec{u} = \vec{u}^\alpha$ in the sense of polynomials on K .
- If $|K \cap \Omega_f(t^\beta)| > 0$ but $|K \cap \Omega_f(t^\alpha)| = 0$, then $\mathbb{E}\vec{u} = \vec{u}^\alpha$ in the sense of polynomials on K out of the largest nearest neighbor cell K_N which fulfills $|K_N \cap \Omega_f(t^\alpha)| > 0$. If two neighbor cells have the same size, the choice is random.

Note that old values of \vec{u}^α extrapolated onto the new domain $\Omega_f(t^\beta)$ will be denoted by $\vec{u}^{\alpha*}$ in the variational formulation. A discrete approximation (u, p) for the solution (u^{n+1}, p^{n+1}) of subproblem (4.1) then can be defined as follows:

Find

$$(\vec{u}, p) \in \mathbb{P}_k(\mathcal{K}_h)^D \times \mathbb{P}_{k-1}(\mathcal{K}_h) =: V_k \quad (4.9a)$$

such that

$$\begin{aligned} \forall (\vec{v}, \tau) \in V_k \\ s\rho_f \left(\frac{3\vec{u}^{n+1}}{2\Delta t}, \vec{v} \right)_{\Omega_f^{n+1}} + c(\vec{w}^{n+1}, \vec{u}^{n+1}, \vec{v}) + b(p^{n+1}, \vec{v}) - a(\vec{u}^{n+1}, \vec{v}) \\ - b(\tau, \vec{u}^{n+1}) = g(\vec{v}, \tau) + p(\vec{u}^{n*}, \vec{u}^{n-1*}, \vec{v}), \end{aligned} \quad (4.9b)$$

with

$$\rho_f \left(\frac{3\vec{u}^{n+1}}{2\Delta t}, \vec{v} \right)_{\Omega_f^{n+1}} = \rho_f \int_{\Omega_f^{n+1}} \frac{3}{2\Delta t} \vec{u}^{n+1} \cdot \vec{v} \, dV. \quad (4.9c)$$

Here the term $c(\vec{w}, \vec{u}, \vec{v})$ denotes the convective term, $b(p, \vec{v})$ the pressure gradient, $a(\vec{u}, \vec{v})$ the viscous terms, $b(\tau, \vec{u})$ the continuity term. The right-hand side (RHS) consists out of a spatial term $g(\vec{v}, \tau)$ and one of the time discretization $p(\vec{u}^{n*}, \vec{u}^{n-1*}, \vec{v})$.

To simplify the notation of the spatial discretization let all domains and faces be part of the time level t^{n+1} , meaning, e.g. $\Omega_f = \Omega_f(t^{n+1})$ and $\Gamma = \Gamma(t^{n+1})$. For discretization a standard symmetric interior penalty (SIP) method (Arnold, 1982) for the viscous terms combined with a Lax-Friedrichs flux (Lax, 1954) for the convective terms is used. The trilinear/bilinear are denoted by

$$\begin{aligned} c(\vec{w}, \vec{u}, \vec{v}) = & -\rho_f \int_{\Omega_f} (\vec{u} \otimes \vec{w}) : \nabla_h \vec{v} \, dV \\ & + \rho_f \oint_{\Gamma} (\{\vec{u} \otimes \vec{w}\} \vec{n}_{\Gamma} + (\lambda/2) \llbracket \vec{u} \rrbracket) \cdot \llbracket \vec{v} \rrbracket \, dS, \end{aligned} \quad (4.10)$$

$$b(p, \vec{v}) = - \int_{\Omega_f} p (\nabla_h \cdot \vec{v}) \, dV - \oint_{\Gamma} \llbracket \vec{v} \rrbracket \cdot \vec{n}_{\Gamma} \{p\} \, dS, \quad (4.11)$$

$$\begin{aligned} a(\vec{u}, \vec{v}) = & -\mu_f \int_{\Omega_f} (\nabla_h \vec{u} : \nabla_h \vec{v}) \, dV \\ & + \mu_f \oint_{\Gamma} \{\nabla_h \vec{u}\} \vec{n}_{\Gamma} \cdot \llbracket \vec{v} \rrbracket \, dS \\ & + \mu_f \oint_{\Gamma} \{\nabla_h \vec{v}\} \vec{n}_{\Gamma} \cdot \llbracket \vec{u} \rrbracket \, dS \\ & - \mu_f \oint_{\Gamma} \eta \llbracket \vec{u} \rrbracket \cdot \llbracket \vec{v} \rrbracket \, dS \end{aligned} \quad (4.12)$$

and

$$b(\tau, \vec{u}) = - \int_{\Omega_f} \vec{u} \cdot (\nabla_h \tau) \, dV - \oint_{\Gamma} \llbracket \tau \rrbracket \vec{n}_{\Gamma} \cdot \{\vec{u}\} \, dS. \quad (4.13)$$

The penalty parameter η is determined by the following equation:

$$\eta = \mu k^2 G_{K_h}, \quad (4.14)$$

where μ is the so called penalty safety factor, k is the polynomial degree and G_{K_h} is a geometrical factor depending on the surface to volume ratio of a particular cut cell. If not stated otherwise, the safety factor is chosen to be 4 in all calculations. More details on the choice of the penalty parameter η and the Lax-Friedrichs parameter λ in (4.10) can be found in the work of Hesthaven and Warburton (2008).

Next, source and boundary condition terms on the right-hand side of the variational problem in (4.9) can be specified. Those can be split into

$$g(\vec{v}, \tau) = q(\vec{v}) + s(\vec{v}) + r(\tau), \quad (4.15)$$

where the functional $q(\vec{v})$ denotes the Dirichlet boundary conditions of the convective operator, $s(\vec{v})$ the volume force and Dirichlet boundary conditions of the linear Stokes problem and $r(\tau)$ the Dirichlet boundary conditions of the continuity equation. Details of the functionals are given by

$$\begin{aligned} q(\vec{v}) = & \rho_f \oint_{\Gamma_D} ((\vec{u}_D \otimes \vec{u}_D) \vec{n}_{\Gamma_D} + (\lambda/2) \vec{u}_D) \cdot \llbracket \vec{v} \rrbracket \, dS \\ & + \rho_f \oint_{\Gamma_j} ((\vec{u}_j \otimes \vec{u}_j) \vec{n}_{\Gamma_j} + (\lambda/2) \vec{u}_j) \cdot \llbracket \vec{v} \rrbracket \, dS \end{aligned} \quad (4.16)$$

$$\begin{aligned} s(\vec{v}) = & - \int_{\Omega_f} \vec{f} \cdot \vec{v} \, dV - \mu_f \oint_{\Gamma_D} \vec{u}_D \cdot (\nabla_h \vec{v} \vec{n}_{\Gamma_D} - \eta \vec{v}) \, dS \\ & - \mu_f \oint_{\Gamma_j} \vec{u}_j \cdot (\nabla_h \vec{v} \vec{n}_{\Gamma_j} - \eta \vec{v}) \, dS, \end{aligned} \quad (4.17)$$

and

$$r(\tau) = \oint_{\Gamma_D} \tau \vec{u}_D \cdot \vec{n}_{\Gamma_D} \, dS + \oint_{\Gamma_j} \tau \vec{u}_j \cdot \vec{n}_{\Gamma_j} \, dS, \quad (4.18)$$

where Γ_D denotes all Dirichlet boundaries out of $\partial\Omega_f \setminus \partial\Omega_j$ and Γ_j denotes the Dirichlet boundaries on $\partial\Omega_j$. Therefore, velocity boundary conditions at the outer boundaries and at the particle interface are treated equally. Note that at this point the actual coupling between fluid and particle interface takes place. Remember, that u_j results out of solving the NEE with calculated hydrodynamical force and torque.

As it can be seen in (4.9b) a time discretization term has to be added to the right-hand side of the system:

$$p(\vec{u}^{n*}, \vec{u}^{n-1*}, \vec{v}) = - \rho_f \int_{\Omega_f^{n+1}} \left(\frac{2\vec{u}^{n*}}{\Delta t} - \frac{\vec{u}^{n-1*}}{2\Delta t} \right) \cdot \vec{v} \, dV. \quad (4.19)$$

Note that again \vec{u}^{n*} and \vec{u}^{n-1*} are fluid velocities at time level t^n resp. t^{n-1} , extrapolated using (4.8) onto the new fluid domain Ω_f^{n+1} and all quantities, including the test functions, are evaluated at time level t^{n+1} .

4.3 Cut cell collision model

Introducing collisions in the present solver, a collision model has to be implemented together with a collision detection approach, which should also work for arbitrary geometries. In contrast, the collision models in the state of the art section of this chapter are mainly based on spherical particles, for which the calculation of distances is trivial. In this section, two collision models will be presented which have been modified in the cut cell DG context for numerical treatment. Both only take collisions with smooth surfaces into account. The question how to treat those models in a discrete setting by using the advantage of cut cells is the main part of this section.

4.3.1 Momentum conservation and repulsive force

The first model is based on the conservation of momentum of rigid bodies colliding, as it is described in Section 2.4. However, in the discrete setting both, the particle-particle and the particle-wall collisions only happen at a certain point in time. To trigger those collisions, a threshold value ρ_1 has been introduced in Section 3.3, which is set to $\rho_1 = 1.5 \sim 2.5h_{min}$, with h_{min} being the minimal grid distance. This threshold leads to the fact that there exist two points of collision at each particle surface, naming \vec{C}_i and \vec{C}_j , see Figure 4.2. Therefore, the particles will not collide for real, meaning \vec{C}_i is never equal to \vec{C}_j . The collision model will be triggered if the distance d_{ij} between two particle is smaller than ρ_1 . After colliding, a boolean value for each particle is set to state the particles already have collided. If the distance between both particles reaches the threshold value again, the booleans are set to false and the particles are allowed to collide again.

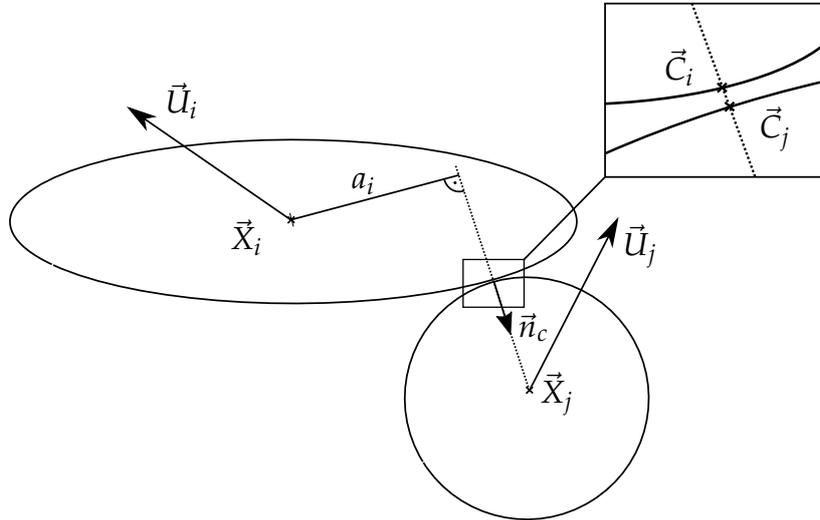


Figure 4.2: Eccentric collision zoomed in point of collision.

If there would be no bool if the particles already collided, they would exchange momentum in each timestep if $d_{ij} < \rho_1$. This results in an oscillatory behavior through pull and push movements between them and finally leads to non-separating particles and thus an error in the collision modeling. In addition, at the timestep of particle-particle collisions the integration of hydrodynamic forces is switched off. This is needed to avoid large pressure oscillations if the interfaces come very close. With this exception, the same procedure is carried out for both particle-particle and particle-wall collisions with momentum conservation.

In addition, a second alternative repulsive force collision model like the one from Glowinski et al. (2001) and Wan and Turek (2006) is implemented for circular shapes. For the repulsive force method, (3.2) and (3.3) are modified avoiding overlap to

$$\vec{F}_{rep} = 0 \text{ if } d_{ij} \geq \rho_1 \quad (4.20a)$$

$$\vec{F}_{\text{rep}} = \frac{1}{\epsilon_p} (\vec{X}_i - \vec{X}_j) (\rho_1 - d_{ij})^2 = \text{if } d_{ij} \leq \rho_1 \quad (4.20b)$$

in the case of particle-particle collisions and

$$\vec{F}_{\text{rep}} = 0 \text{ if } d'_i > \rho_1 \quad (4.21a)$$

$$\vec{F}_{\text{rep}} = \frac{1}{\epsilon_w} (\vec{X}_i - \vec{X}'_i) (\rho_1 - d'_i)^2 = \text{if } \rho_1 \geq d'_i \geq \rho_2 \quad (4.21b)$$

$$\vec{F}_{\text{rep}} = \frac{1}{\epsilon'_w} (\vec{X}_i - \vec{X}'_i) (\rho_1 - d'_i) \text{ if } d'_i \leq \rho_2 \quad (4.21c)$$

for particle-wall collisions. Here, again $\rho_1 = 1.5 \sim 2.5h_{\min}$ and $\rho_1 > \rho_2 > 0$, meaning no overlap is present. The stiffness parameters are set to

$$\epsilon_p = \epsilon_w = \frac{\rho_1^2}{2} \quad \text{and} \quad \epsilon_{w'} = \frac{\rho_1}{2}. \quad (4.22)$$

In contrast to the conservative collision model, the repulsive force model is smooth in time. Meaning there is no particular point in time where the particles collide and no boolean value is needed. Both collision models are compared with results from literature for the well known draft, kissing and tumbling testcase in Subsection 5.1.5.

4.3.2 Collision detection based on cut cells

As it is mentioned in Section 3.3, it is inefficient to check distances for every possible particle collision pair. Therefore, a collision detection based on the cut cell approach is introduced.

The starting point of the collision model is the fact, that every particle immersed in the fluid already knows its cut cells by evaluating a function representing the shape of the geometry and therefore geometrical information on the location of the particle is present. Let $\mathcal{K}_C^i = \{K_{1C}^i, \dots, K_{n_C}^i\}$ be the set of all cut cells representing the i -th particle. Since for every cell there is also information about cell neighbors, a subset of all cut cell neighbors \mathcal{K}_N^i of those cells can be created for particle i and j , see Figures 4.3a and 4.3b. With this, the intersection of both sets can be determined in Figure 4.3c and saved to a set \mathcal{K}_h^{ij} . Finally, the neighbors of this intersection cells are collected in a subset, if they are cut cells. For those cells, which can be seen in Figure 4.3d, a detailed distance calculation can be performed.

Therefore, the closest distances between two arbitrary shaped particles has to be evaluated. Here, only points which are located on the interface are used and their distance to each other is calculated. Note, that this can be very computational expensive, if interface points in all cut cells of particle i are compared with points in all cut cells of the j -th particle. As already mentioned, in order to minimize the computational cost, only cut cells of the i -th particle are considered, which are neighbors of the intersection set \mathcal{K}_h^{ij} . This method renders to be very efficient for a small number of particles with

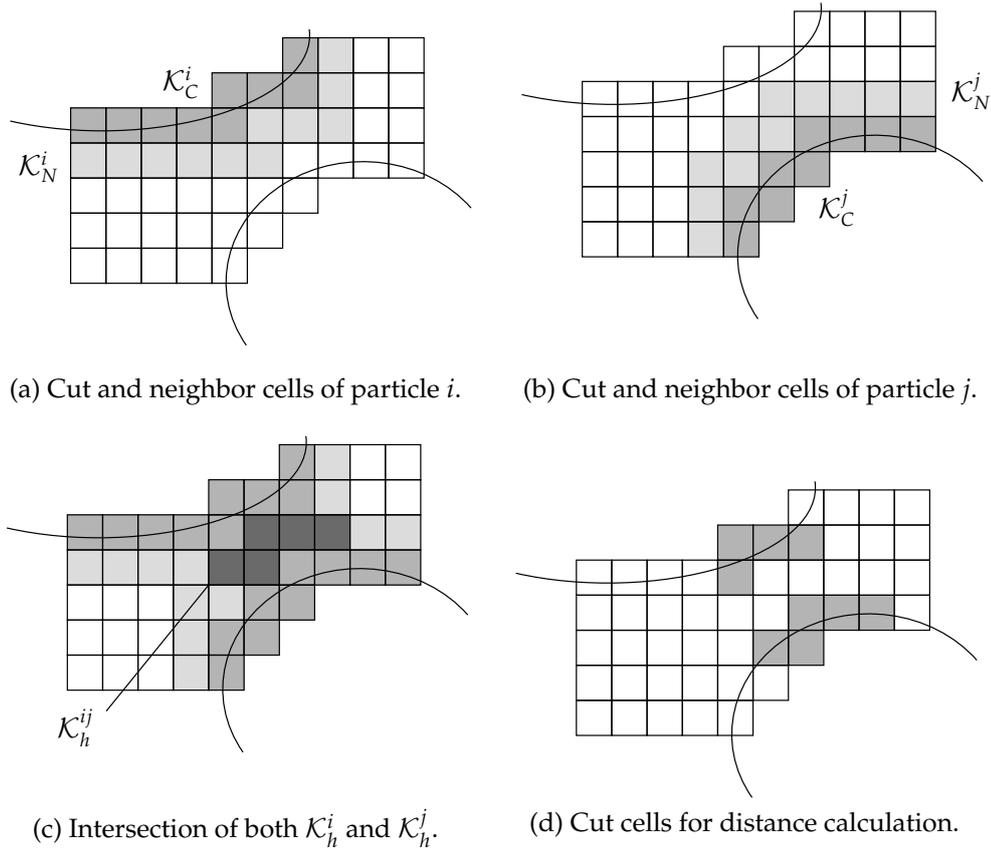


Figure 4.3: Collision detection based on cut cells for arbitrary geometry.

arbitrary shape, whereas there exist better methods for large number of circular or spherical particles, e.g. see Sigurgeirsson et al. (2001).

In Figure 4.4, the determination of interface points for a particular cut cell can be seen. For each quadrature node in every cut cell of particle i and j the closest point on the interface is determined and saved. It is important to state, that Figure 4.4 is only a schematic representation and the number of interface points increases massively with the quadrature order. Of course, this has an impact on the distance calculation and the direction of the collision. Therefore, the choice of a sufficient high quadrature order is important especially for arbitrary shaped particles.

In total, the complete scheme proceeds as follows:

1. Determine cut cells \mathcal{K}_C^i and neighbors of cut cells \mathcal{K}_N^i belonging to the i -th particle.
2. Take the union of all cut cells and neighboring cells for every i -th particle $\mathcal{K}_h^i = \mathcal{K}_C^i \cup \mathcal{K}_N^i$.
3. For every pair of particles ij in the computational domain, create a set of all cells \mathcal{K}_h^{ij} , which are contained in both particle cells $\mathcal{K}_h^{ij} = \mathcal{K}_h^i \cap \mathcal{K}_h^j$.

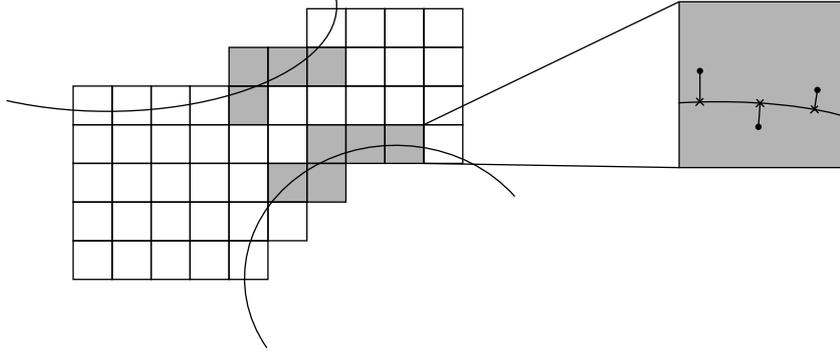


Figure 4.4: Determination of interface points inside a cut cell.

4. If $\mathcal{K}_h^{ij} \neq \emptyset$ the distance calculation can be started, otherwise the particles are not close.
5. For distance calculation, points at the interface on the closest cut cells are compared to yield the shortest distance d_{ij} between particle i and j .
6. If $d_{ij} < \rho_1$, the collision model is triggered.

For particle-wall collisions the approach is very similar. If a cell of \mathcal{K}_h^i contains a boundary edge of the computational domain, the distance calculation is triggered. For distance calculation, points on the interface are compared to vertices at the boundary edges of all boundary edges in \mathcal{K}_h^i .

4.4 Integration and agglomeration on cut cells

The crucial part about a cut cell immersed boundary method is the accurate and efficient integration. In the discretization the integrals

$$\begin{aligned} \oint_{\Gamma_I} f \, dS & \quad (\text{inner edges between cut cells}), \\ \oint_{\Gamma_j} f \, dS & \quad (\text{particle surface}), \end{aligned}$$

and

$$\oint_{K \cap \Omega_f} f \, dV \quad (\text{cut cell volumes})$$

have to be computed numerically on a reference cell $K \subset \mathbb{R}^D$, where f has to be sufficiently smooth. For the solver the method of HMF integration proposed by Müller et al. (2013) is used. The key point is the construction of HMF rules using the Gauss theorem

$$\int_{K \cap \Omega_f} \nabla \cdot \vec{f} \, dV - \oint_{K \cap \Gamma_j} \vec{f} \cdot \vec{n}_{\Gamma_j} \, dS = \oint_{K \cap \Gamma_I} \vec{f} \cdot \vec{n}_{\Gamma_I} \, dS \quad (4.23)$$

for a given vector field \vec{f} . Kummer (2016) introduced a variant of the original HMF method which does not only use the Gauss theorem but also enforces the Stokes theorem which reads

$$\oint_{K \cap \Gamma_j} \kappa \vec{n}_{\Gamma_j} \cdot \vec{f} - (I - \vec{n}_{\Gamma_j} \otimes \vec{n}_{\Gamma_j}) : \nabla \vec{f} \, dS = - \int_{\Gamma_I \cap \Gamma_j} \vec{t} \cdot \vec{f} \, dL, \quad (4.24)$$

where the integral on the RHS denotes a zero-dimensional point measure over all points in $\Gamma_I \cap \Gamma_j$, κ is the curvature of the interface and \vec{t} is the outward tangent to Γ_j .

For numerical integration the integrals

$$\int_{(\vec{x}, \underline{w})}^{num} f := \sum_{i=1}^L w_i f(\vec{x}_i) \quad (4.25)$$

are approximated with weights w_i on a set of nodes \vec{x}_i . The main idea of this approach is to compute the least-square solution of the following system in order to get the weights \underline{w}^{Γ_j} and \underline{w}^{Ω_f} for given nodes \vec{x} over particle surfaces Γ_j and cut cell volumes $\Omega_f \cap K$:

$$\begin{aligned} \int_{(\vec{x}, \underline{w}^{\Omega_f})}^{num} \nabla \cdot \vec{f} - \oint_{(\vec{x}, \underline{w}^{\Gamma_j})}^{num} \vec{f} \cdot \vec{n}_{\Gamma_j} &= \oint_{\Gamma_j} \vec{f} \cdot \vec{n}_{\Gamma_j} \, dS \\ \oint_{(\vec{x}, \underline{w}^{\Gamma_j})}^{num} \kappa \vec{n}_{\Gamma_j} \cdot \vec{f} - (I - \vec{n}_{\Gamma_j} \otimes \vec{n}_{\Gamma_j}) : \nabla \vec{f} \, dS &= - \int_{\Gamma_I \cap \Gamma_j} \vec{t} \cdot \vec{f} \, dL \end{aligned} \quad (4.26)$$

$$\forall \vec{f} \in \mathbb{P}_{k'}(\mathcal{K}_h)^D$$

with surface weights \underline{w}^{Γ_j} and volume weights \underline{w}^{Ω_f} . Note that for a sufficient high order quadrature in this work, the relation between polynomial space and quadrature order is chosen to be $k' = 3k$ for surface and volume integrals in cells and $k' = 3k + 2$ for the particle surfaces. Further, a final comment on the least-square solution is made: The method of HMF always leads to under-determined systems and therefore the global solution with the smallest norm-value is being chosen. Further details and variants of the HMF can be extracted from the references cited above.

Since cut cells can become arbitrary small a procedure called cell agglomeration is used to restrict the condition number of the system (Kummer, 2016; Müller et al., 2016). Here, all cells containing fluid under a particular threshold value ($0 \leq \gamma < 1$) are agglomerated to their nearest neighbor. For all calculations in this paper, a cell agglomeration threshold of $\gamma = 0.2$ is used. For a schematic representation of the cell agglomeration see Figure 4.5. The details of the agglomeration algorithm are not important in this context, but can be extracted from Kummer (2016).

4.5 Conclusion

This chapter focuses on the method developed in this work. Starting with general discretization aspects, leading to the question of how boundary conditions at the

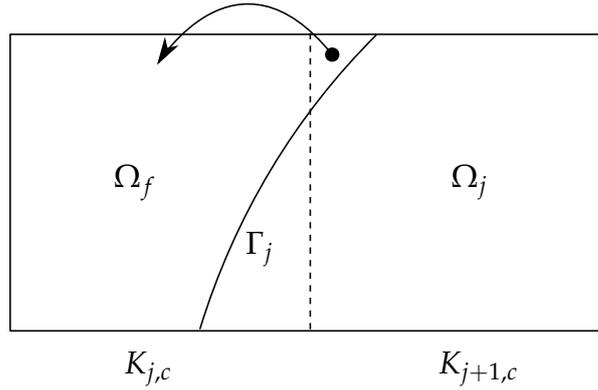


Figure 4.5: A small fluid cut cell $K_{j+1,c}$ is agglomerated to its nearest neighbor $K_{j,c}$, because the agglomeration factor of fluid inside the cut cell is below a certain threshold of $\gamma = 0.2$

particle interfaces are imposed in the cut cell DG framework. Additionally, the time discretization and coupling technique is described, followed by a presentation of the HMF integration strategy on cut cells. For flows containing more than one particle, two possible collision models are presented. In order to detect possible collisions, a collision detection algorithm based on the relation between cut cells and a distance calculation at the interface is also presented.

In the Chapter 5, several numerical results are presented. All of them show the ability of the method to save DoF with a higher-order approach by simultaneously gaining the same accuracy. Also a comparison of physical quantities between the non-conservative splitting and conservative moving interface approach is made for the fully coupled example of a single disk falling in a channel Subsection 5.1.4. Furthermore, although the method is presented in case of two dimensions, an extension to three dimensional cases is straightforward and the same ability will be proven in Section 5.3.

5 Numerical results

In this chapter the presented cut cell DG method for particulate flows is validated using various test calculations with increasing complexity. During this process several features are tested separately. Those are always followed by a more complicated test, which couples them together. A simple draft of the overall procedure can be seen in Figure 5.1.

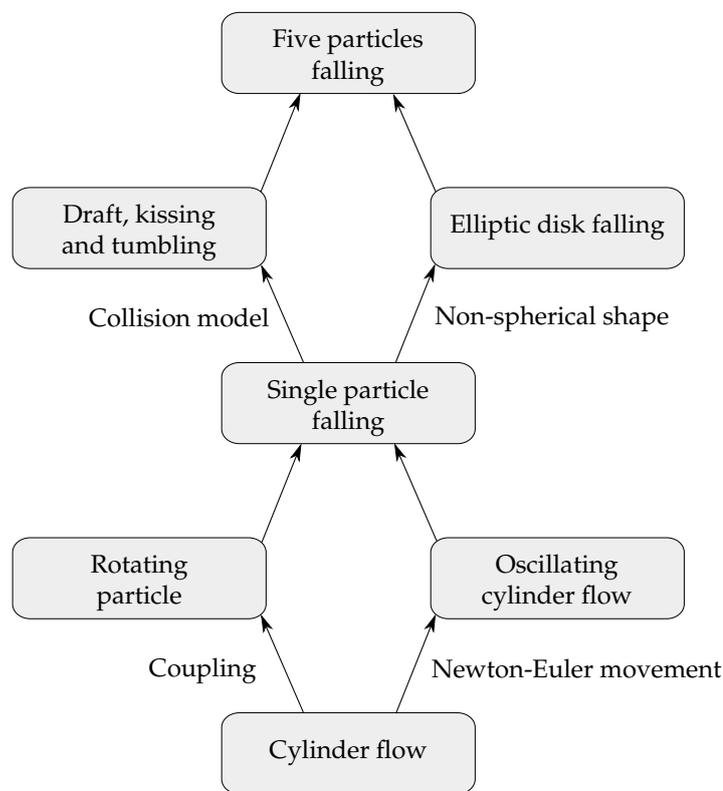


Figure 5.1: Overview of numerical results with increasing complexity discussed within Chapter 5.

The considerations start with pure immersed boundary calculations in Subsection 5.1.1, followed by evaluation of the coupling in direction from particle to fluid and vice versa in Subsection 5.1.2 and Subsection 5.1.3. In Subsection 5.1.4, the first fully coupled test for a single particle is investigated. All the previous mentioned tests are also part of the publication from Krause and Kummer (2017). Furthermore, the publication mentioned above is extended. For this, the proposed cut cell collision models from Section 4.3 are applied for two falling circular particles in a channel in Subsection 5.1.5. After this, the shape of the particle in Section 5.2 is modified to an ellipse and other shapes to show the ability of the method to handle complex geometries. In Subsection 5.2.3, all features are coupled together by the simulation of five particles falling in fluid. Before

concluding this chapter, in Section 5.3 the method is extended to three dimensional problems with non-moving spherical particles to proof the ability of the method for three dimensions. If not stated otherwise, the splitting approach is used for the time discretization of moving boundaries.

5.1 Flow around particles with circular shape

First, particles with circular shape are focused. Most methods in literature assume particles to be spherical. Therefore, the validation of this method is started with the same assumption. Moreover, hydrodynamic forces are easier integrated and centric collisions are better handled because the distance between surface and the center of mass is the same for all surface points.

5.1.1 Flow around a fixed cylinder¹

In the beginning, a steady test case for a laminar flow around a fixed cylinder at a Reynolds number of 20 is analyzed. The test case has benchmark character and was published by Schäfer et al. (1996). The geometry is a channel flow with channel height $H = 0.41$ and length $L = 2.2$. The cylinder is placed at $(0.2, 0.2)$ and has a diameter of $D = 0.1$. Figure 5.2 shows the computational domain which was taken from the reference cited above.

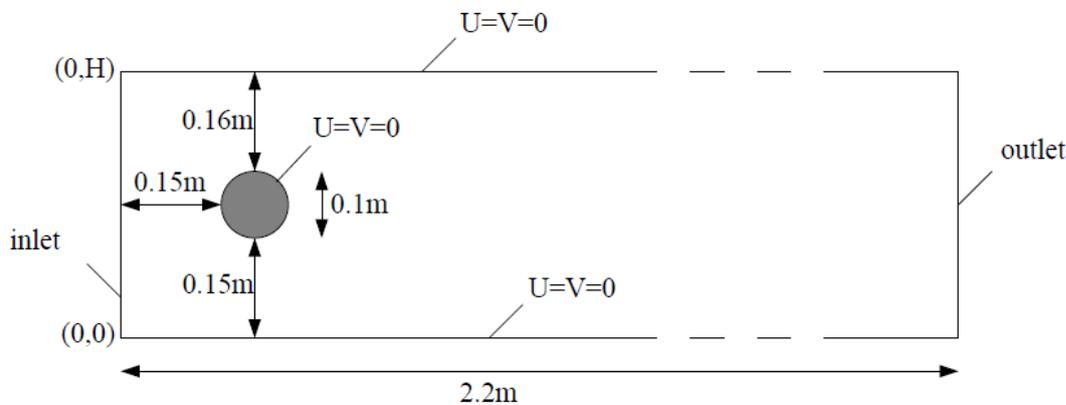


Figure 5.2: Computational domain for cylinder flow taken from Schäfer et al. (1996).

The boundary conditions are denoted as follows: At the left wall a velocity inlet with a parabolic velocity profile of $U(y) = 6y(H - y)/H^2$ and $V = 0$ is applied. At the cylinder surface as well as at the upper and lower walls of the channel no-slip boundary conditions are imposed. As outflow boundary condition a common pressure outlet is used.

The results of the calculations with three different penalty safety factors μ , see (4.14), in combination with polynomial degrees $k = 1, 2, 3$ may be taken from Table 5.1. The second column describes the polynomial degree of the test functions k , in the third

¹Modified version of (Krause and Kummer (2017), Section 4.1)

one there are the total spatial DoF, followed by C_D for drag- and C_L for lift-coefficient, which are denoted through the following equations:

$$C_D = \frac{2F_D}{\rho_f \bar{U}^2 D} \quad (5.1a)$$

and

$$C_L = \frac{2F_L}{\rho_f \bar{U}^2 D} \quad (5.1b)$$

with F_D being the drag force, ρ_f being the density of the fluid phase, D being the diameter of the cylinder and \bar{U} being the spatially averaged velocity at the inlet. The deviation of the calculated values from the mean values of Schäfer et al. (1996) is also given in brackets.

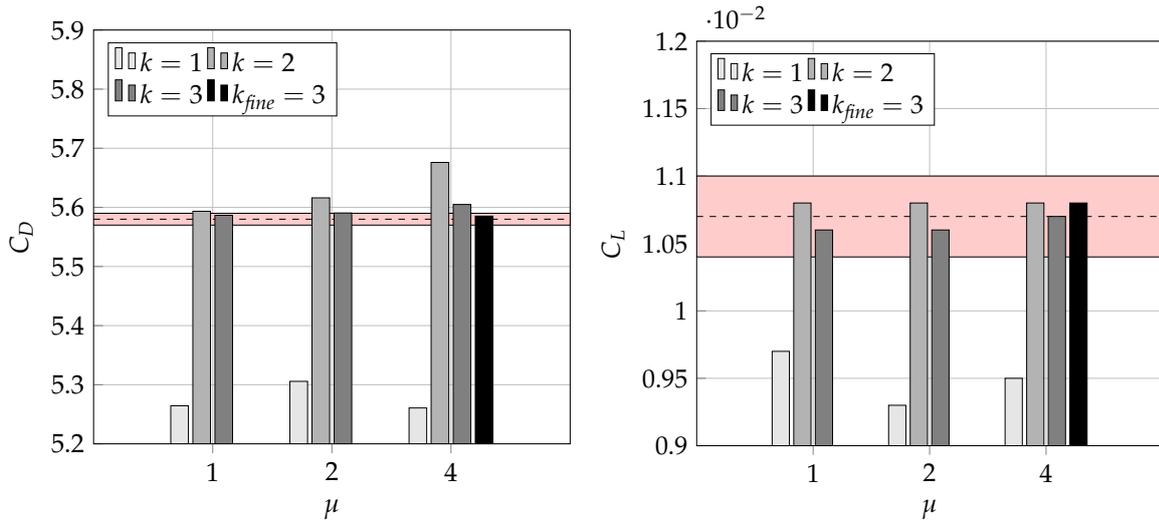
Table 5.1: Results for steady flow around a fixed cylinder at Re=20.

μ	k	DoF	C_D (\pm Tol.)		C_L (\pm Tol.)	
1	1	46 000	5.2644	(+ 5.7 %)	0.0097	(+ 9.4 %)
	2	45 000	5.5934	(− 0.2 %)	0.0108	(− 0.9 %)
	3	46 000	5.5866	(− 0.1 %)	0.0106	(+ 0.9 %)
2	1	46 000	5.3057	(+ 4.9 %)	0.0093	(+ 13.1 %)
	2	45 000	5.6161	(− 0.7 %)	0.0108	(− 0.9 %)
	3	46 000	5.5905	(− 0.2 %)	0.0106	(+ 0.9 %)
4	1	46 000	5.2607	(+ 5.7 %)	0.0095	(+ 11.2 %)
	2	45 000	5.6760	(− 1.7 %)	0.0108	(− 0.9 %)
	3	46 000	5.6049	(− 0.5 %)	0.0107	(\pm 0.0 %)
	3	120 000	5.5851	(− 0.1 %)	0.0108	(− 0.9 %)
Schäfer et al. (1996)			5.5800	(\pm 0.2 %)	0.0107	(\pm 3 %)

It can be seen, that for the choice of all penalty safety factors better agreement with the literature is yield if the polynomial degree is increased and background mesh is coarsened simultaneously. This can also be confirmed by Figure 5.3. Here the mean value is represented as dashed line and the tolerance of the mean value is shown by the shaded area.

It should be further mentioned that the penalty has an impact on drag and lift values especially if the general number of DoF is low for small polynomial degrees k . In Table 5.1 also one calculation with an increased number of DoF is presented and literature values are perfectly reached with a penalty safety factor of $\mu = 4$. Those results can also be seen in Figure 5.3 indicated by black bars.

Next, the testcase is extended to an unsteady solution by increasing Re=100. The height of the channel is again chosen to $H = 0.41$ in order to accelerate the beginning of vortex shedding, because the flow field at the cylinder position results to be slightly asymmetric.



(a) Impact of penalty factor on Drag coef.

(b) Impact of penalty factor on Lift coef.

Figure 5.3: Results of different penalty factors combined with different polynomial degrees for Reynolds number (Re)=20. The gray area denotes the error bounds given in the publication of Schäfer et al. (1996).

As it can be seen in Table 5.2, different choices of the polynomial degree were tested. In Table 5.2 $(C_D)_{max}$ is the maximum drag, $(C_L)_{max}$ the maximum lift coefficient and

$$St = \frac{f_n \cdot D}{u} \quad (5.2)$$

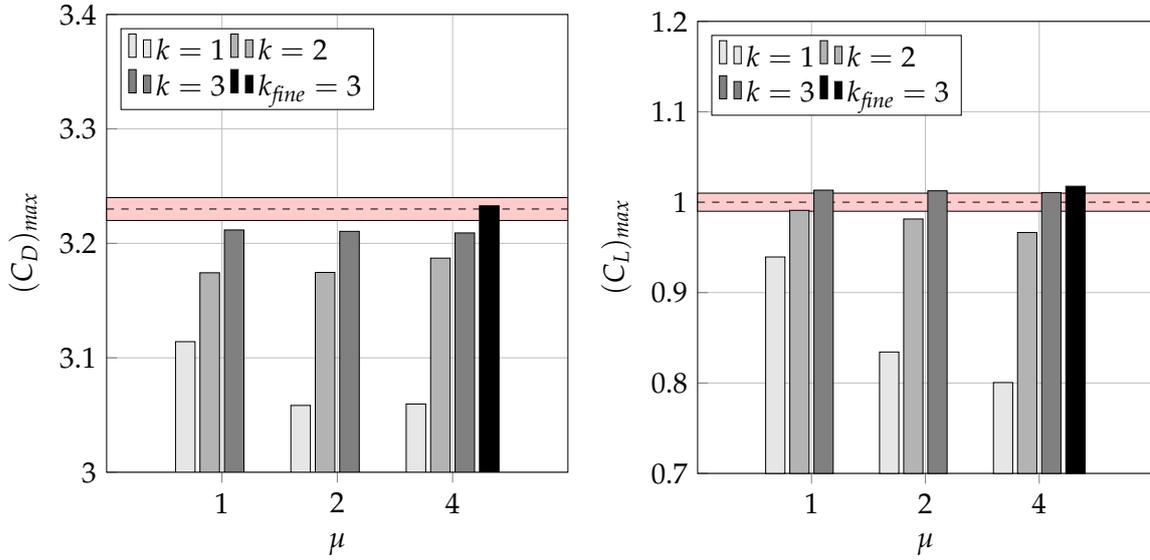
is the Strouhal number (St). In (5.2) f_n is the natural shedding frequency and u the flow velocity.

Table 5.2: Results for unsteady flow around a fixed cylinder at $Re=100$.

Δt	μ	k	DoF	$(C_D)_{max}$ (\pm Tol.)	$(C_L)_{max}$ (\pm Tol.)	St (\pm Tol.)
0.05	1	1	46 000	3.1141 (+ 3.6 %)	0.9394 (+ 6.1 %)	0.2941 (+ 2.0 %)
		2	45 000	3.1743 (+ 1.7 %)	0.9910 (+ 0.9 %)	0.2985 (+ 0.5 %)
		3	46 000	3.2117 (+ 0.6 %)	1.0133 (− 1.3 %)	0.3030 (− 0.1 %)
0.05	2	1	46 000	3.0584 (+ 5.3 %)	0.8343 (+ 16.6 %)	0.2985 (+ 0.5 %)
		2	45 000	3.1746 (+ 1.7 %)	0.9814 (+ 1.9 %)	0.2985 (+ 0.5 %)
		3	46 000	3.2105 (+ 0.6 %)	1.0126 (− 1.3 %)	0.3030 (− 0.1 %)
0.05	4	1	46 000	3.0596 (+ 5.3 %)	0.8005 (+ 20.1 %)	0.2985 (+ 0.5 %)
		2	45 000	3.1871 (+ 1.3 %)	0.9665 (+ 3.4 %)	0.2985 (+ 0.5 %)
		3	46 000	3.2091 (+ 0.7 %)	1.0107 (− 1.1 %)	0.3030 (− 0.1 %)
		3	120 000	3.2328 (+ 0.1 %)	1.0176 (− 1.8 %)	0.3030 (− 0.1 %)
Schäfer et al. (1996)				3.2300 (\pm 0.3 %)	1.0000 (\pm 1.0 %)	0.3000 (\pm 2.0 %)

It can be confirmed again, with a fixed penalty safety factor, increasing the polynomial degree and keeping the total number of DoF fixed, a much better result is obtained if it is compared to the benchmark data of Schäfer et al. (1996), see also Figure 5.4.

The cited results were obtained by different academic and commercial codes which all use a conventional body-fitted grid. Nevertheless, the penalty factor has a particular influence on drag and lift forces due to the penalization of the values at the particle surface. In contrast to the calculation at $Re=20$ the grid refinement only leads to the drag coefficient which is closer to the literature whereas the lift coefficient is slightly deteriorated.



(a) Impact of penalty factor on Drag coef.

(b) Impact of penalty factor on Lift coef.

Figure 5.4: Results of different penalty factors combined with different polynomial degrees for $Re=100$. The gray area denotes the error bounds given in the publication of Schäfer et al. (1996).

In consequence, those first test cases show the ability of the method to handle a body immersed in the grid and an accurate integration of the total stresses for obtaining the hydrodynamic forces. However, these forces are only computed for validation purposes and the choice of the penalty safety factor has also an influence on the results, especially if the resolution at the surface is coarse the derived drag and lift values become very sensitive. In the following, all calculations are performed with a penalty safety factor of $\mu = 4$ due to the recommendation made by Kummer (2016) in order to reach stability of the scheme.

5.1.2 Flow around a transversally oscillating cylinder²

In this test case the flow around a cylinder oscillating in a free stream is investigated. The motion is prescribed by a given function $\vec{X}_c = (x_c(t), y_c(t))$. The total motion can be considered as a free flow with a superimposed oscillating flow part. The oscillating flow field is induced by

$$\rho_f \frac{d\vec{u}_c}{dt} = -\nabla p \quad \text{in } \Omega_f, \quad (5.3)$$

²Modified version of (Krause and Kummer (2017), Section 4.2)

with \vec{u}_c being the time dependent oscillation velocity.

As a result of this summation, the force \vec{F}_j acting on the cylinder has contributions from the force \vec{F}_{FL} resulting from the free flow and the Froude-Krylov force \vec{F}_{FK} resulting from the unsteady pressure field generated by the oscillations, see (5.4). In order to compare our lift coefficient with literature values, where the cylinder is fixed and the velocity boundary conditions are oscillating, the total hydrodynamic force, which is calculated by solving (2.8a), \vec{F}_{FK} has to be subtracted. This technique is well-known and was used by Meneghini and Bearman (1995).

$$\vec{F}_j = \vec{F}_{FL} + \vec{F}_{FK}, \quad (5.4)$$

where

$$\begin{aligned} \vec{F}_{FK} &= - \int_{\partial\Omega_j} p \cdot \vec{n} \, ds \stackrel{\text{Gauss}}{=} - \int_{\Omega_j} \nabla p \, dV \stackrel{(5.3)}{=} \rho_f \int_{\Omega_j} \frac{d\vec{u}_c}{dt} \, dV \\ &= \rho_f V_j \frac{d^2 \vec{X}_c}{dt^2}. \end{aligned} \quad (5.5)$$

The geometrical data of this test case was first described by Lai and Peskin (2000) and later used by Uhlmann (2005). A cylinder with a diameter of $d = 0.3$ is placed at the origin of the computational domain $\Omega = [-1.85, 6.15] \times [-4, 4]$. At $x = -1.85$ a velocity inlet condition with a uniform free stream velocity of $\vec{u} = (1, 0)$ is imposed. A pressure outlet condition for all three other outer boundaries is used. The cylinder surface is treated by using velocity boundary conditions of the oscillating motion and $\text{Re} = 185$. Furthermore, the cylinder is forced to oscillate only in y -direction by describing its position using the function

$$y_c(t) = 0.2 d \cos(2\pi f_f t), \quad (5.6)$$

with d being the diameter of the cylinder and f_f being 0.8 times the natural shedding frequency f_n at $\text{Re} = 185$ in case of a non-oscillating cylinder. The domain is discretized using a mesh with hanging nodes in order to obtain a fine resolution near the cylinder surface without increasing the total number of DoF massively. In order to keep the total number of DoF fixed for different polynomial degrees three meshes are created. The coarsest background mesh for the calculation using a polynomial degree of 3 for velocity and 2 for pressure can be seen in Figure 5.5.

Results of these calculations can be seen in Table 5.3. Here the mean drag coefficient $\overline{C_D}$, the amplitude of the drag coefficient C'_D and the root-mean-square value of the lift coefficient $(C_L)_{rms}$ are compared with calculations of Uhlmann (2005) (immersed boundary method) and Lu and Dalton (1996) (body-fitted grid). Additionally, Uhlmann (2005) implemented in his work the forcing scheme of Kajishima and Takiguchi (2002).

It can be seen, that the results are in good agreement with those calculations taken from literature. The best agreement is yield if the results are compared to those with the forcing scheme of Kajishima et al. (2001), Kajishima and Takiguchi (2002). In their

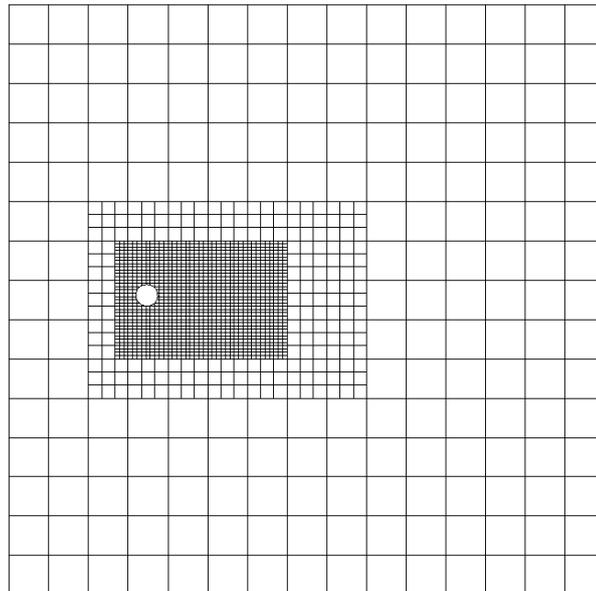


Figure 5.5: Background mesh at $t = t_0$ for calculation with polynomial degree of $k = 3$. For the finest cells around the cylinder a resolution of $d/h=5$, with d being the cylinder diameter.

Table 5.3: Results for unsteady flow around an oscillating cylinder at $Re=185$ with $\Delta t = 0.1$ for C_D and C_L .

Method	k	DoF	$\overline{C_D}$	C'_D	$(C_L)_{rms}$
Cut Cell DG	1	61 000	1.24	± 0.067	0.247
	2	55 000	1.27	± 0.057	0.235
	3	47 000	1.26	± 0.078	0.225
Uhlmann (2005)			1.380	± 0.063	0.176
Uhlmann (2005) with Kajishima and Takiguchi (2002) forcing			1.282	± 0.088	0.223
Lu and Dalton (1996)			1.25		0.18

work they use a volume integration over the so-called interacting force which is a linear interpolation between the fluid velocity and the particle velocity, also including cells which are cut by the interface using a volume fraction approach.

In contrast to the observation made by Uhlmann (2005) using the method of Kajishima et al. (2001) strong oscillations cannot be observed in the drag hysteresis like in the publication of Uhlmann. The drag hysteresis of the calculation, using a polynomial degree of $k = 3$ and a timestep of $\Delta t = 0.1$, can be seen in Figure 5.6. This is probably because a coarser background mesh discretization is used with a high polynomial degree, naming $d/h = 5$ for $k = 3$ in comparison with Uhlmann $d/h = 38.4$, where h is the mesh width. Also the use of an implicit time stepping scheme and a larger timestep, i.e. 64 timesteps per cylinder oscillation, certainly has an influence.

Nevertheless, the rms-value of the lift coefficient is predicted much higher than in the reference calculation of Lu and Dalton (1996). This finding is not observed by the immersed boundary method of Uhlmann. To sum up, the best agreement with literature values can be obtained using a small amount of DoF at a higher-order DG approximation. Therefore, the solver is able to handle fixed motion within the grid accurately.

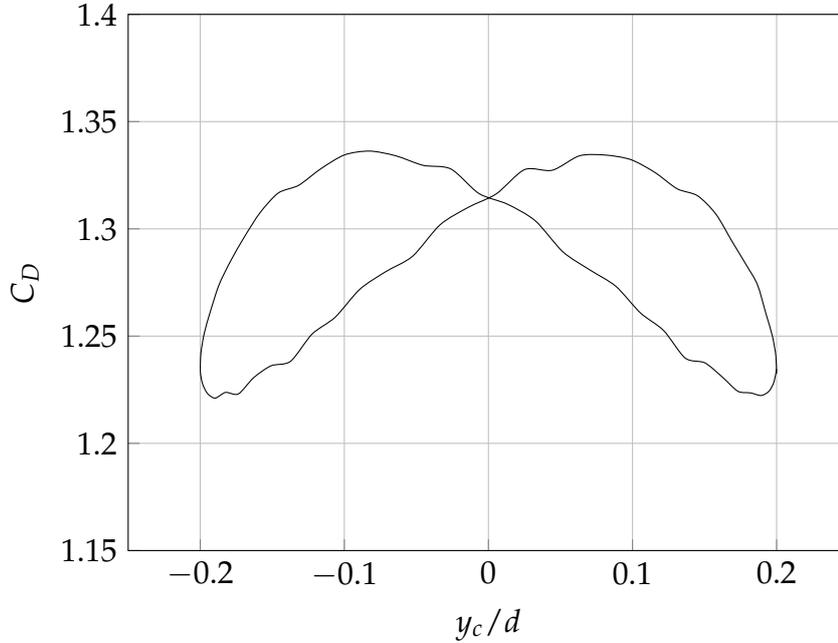


Figure 5.6: Drag hysteresis for one cylinder oscillation.

5.1.3 One rotating particle in a Couette flow³

The coupling between cylinder and fluid is first tested in a fixed domain environment which was described by Wan and Turek (2006). Therefore, a particle is placed between two walls, which move in opposite directions with a velocity $v_y = 0.02$ and $v_y = -0.02$, creating a linear shear flow in between the walls. The channel height is $H = 6$, its width is $W = 4$ and the particle center is placed at position $\vec{X} = [2, 3]$. Top and bottom of the computational domain are bounded by using periodic boundary conditions in y -direction. The density of the particle and the fluid are chosen equally to $\rho_f = \rho_p = 1$. The viscosity of the fluid is denoted by $\mu_f = 0.01$.

The particle is fixed but can rotate around its center according to the resulting torque from the hydrodynamical forces. Initially, the particle as well as the fluid are at rest. If the radius of the particle is small enough the angular velocity of the particle should be close to $U_p = 0.005$ according to the vorticity field in the linear shear flow while reaching the steady state solution much faster with decreasing radius R . Different particle radii are used and the terminal angular velocity is computed. At the same time a polynomial degree of $k = 3$ for velocity is used and a time step size of $\Delta t = 1$

³Modified version of (Krause and Kummer (2017), Section 4.3)

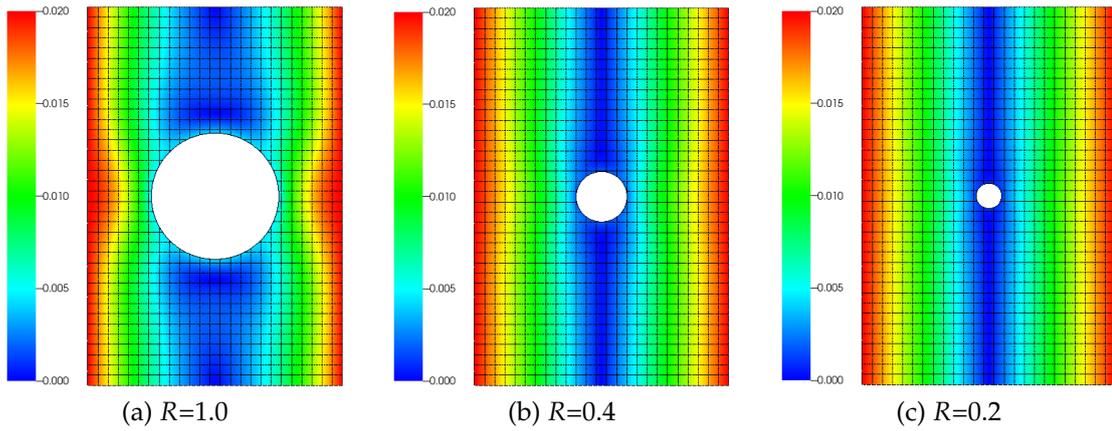


Figure 5.7: Velocity magnitude in the steady state for different radii with background mesh.

for $R = 1$, $\Delta t = 0.5$ for $R = 0.4$ and $\Delta t = 0.25$ for $R = 2$. In total the system for the smallest radius has around 25 000 DoF in space, whereas Wan and Turek (2006) use 710 000 DoF. The values obtained are compared with the results of Wan and Turek (2006) in Table 5.4.

Table 5.4: Results of a particle in shear flow.

Particle radius R	Terminal angular velocity ω_j	
	Cut Cell DG	Wan and Turek (2006)
1	0.0042972	0.0043148
0.4	0.0049177	0.0048697
0.2	0.0049488	0.0049584

It can be seen that our results are in good agreement with the reference cited, however there is only one literature source for this type of flow. Those results also have never been compared to experiments. Therefore, Table 5.4 only compares both results. Nevertheless, the predicted physical behavior of the increasing terminal angular velocity with shrinking diameter has been shown. A plot of the velocity magnitude and the background mesh for all three calculations can be seen in Figure 5.7. In addition, the angular velocity over time is plotted in Figure 5.8 to show the second prediction of less time needed to reach the steady state with decreasing radius. Thus, it can be seen that the smaller the diameter the faster the terminal angular velocity of the particle is reached.

A modification of the present test case is also used to carry out a mesh size convergence study. The particle radius is chosen to be $R = 1$ and the coupling is turned off, such that a no slip boundary condition is imposed at the particle surface. Therefore, a steady solution in time is obtained again. For each polynomial degree k for velocity and $k' = k - 1$ for pressure a reference solution is calculated by using a very fine grid. Then four calculations for each polynomial degree were carried out and the L^2 -Error

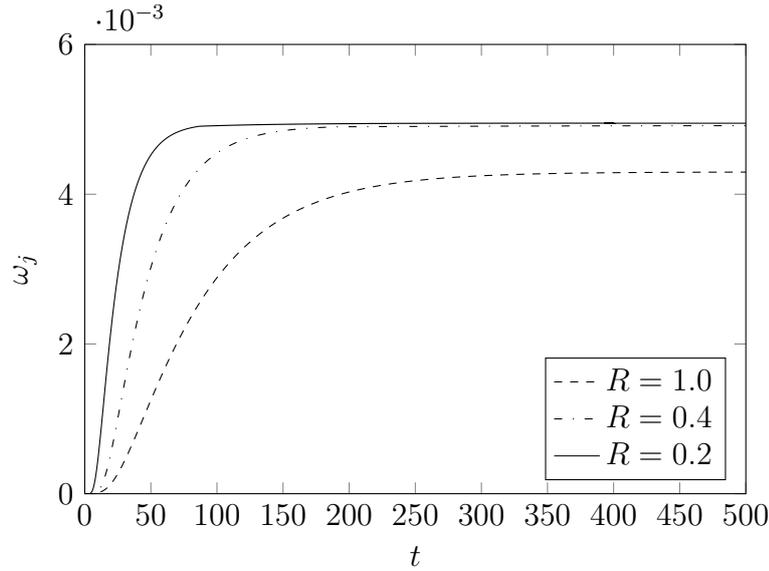


Figure 5.8: Angular velocity of particle with different radii over time.

in comparison with the aforementioned reference solution is determined. In Figure 5.9 the convergence behavior can be seen. Expected decreasing functions can be observed with the exception of the coarsest mesh for the pressure at polynomial degree of $k = 1$. This is due to the fact that for the calculation mentioned above the mesh is under resolved.

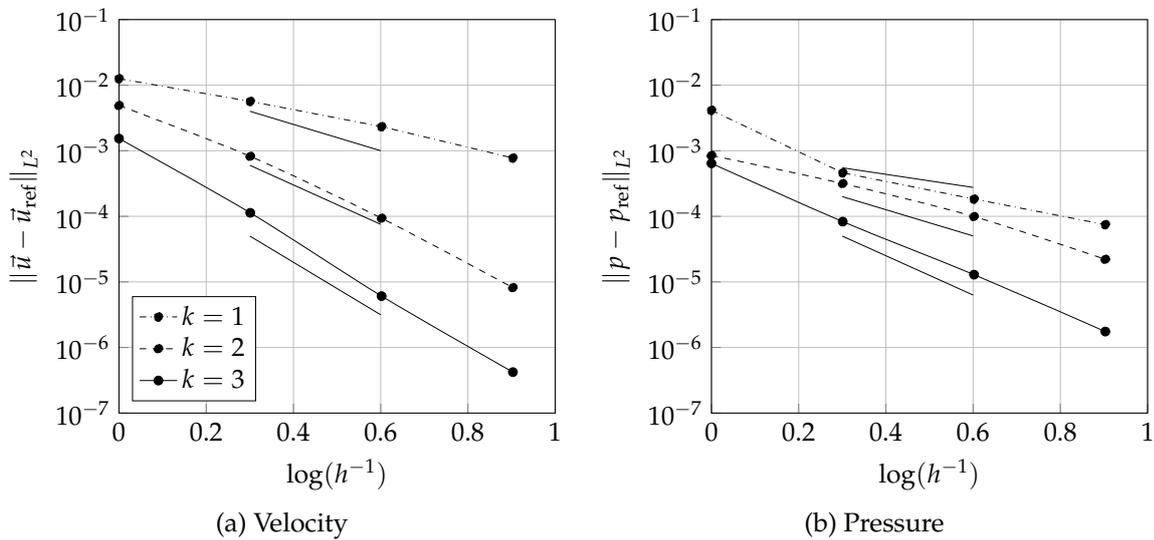


Figure 5.9: hp-convergence for velocity and pressure. For each polynomial degree the ideal slope is shown.

An experimental order of convergence (EOC) of $k + 1$ for velocity and k for pressure is expected. Because of the under resolved calculations for the coarsest mesh size it was neglected in determining the EOC in Table 5.5. All in all, good agreement with

the expectations can be seen and therefore, the reliability of the method in terms of hp-refinement is proven.

Table 5.5: Experimental order of h-convergence.

k/k'	EOC velocity	EOC pressure
1/0	1.4	1.3
2/1	3.3	1.9
3/2	4.0	2.8

5.1.4 Single particle falling in incompressible fluid⁴

The next test case is calculated in order to proof the ability of the solver to handle coupled particle motion within a fluid with moving domains. Therefore, a single disk falling in an incompressible fluid due to gravity force is chosen. This test was studied by Wan and Turek (2006).

The dimensions of the computational domain are $W = 2$ and $H = 6$ and the particle with diameter of $d = 0.25$ is centered at $X_p = (1, 4)$ at time $t = t_0$. The density ratio of the particle and fluid is denoted with $\frac{\rho_j}{\rho_f} = 1.25$. For validation purposes the particle Reynolds number is calculated every timestep using $Re_p = |\vec{U}| \cdot d \cdot \rho_j / \nu$, with \vec{U} being the particle velocity. Again, calculations have been conducted for three different polynomial degrees, keeping the total number of DoF almost fixed. The background mesh for the calculation with $k = 3$ can be seen in Figure 5.10 at initial time. The mesh is equidistant in the whole computational domain and the resolution is around four cells in direction of the particle diameter.

Table 5.6: Results single particle falling in incompressible fluid.

Method	Δt	k	DoF	max Re_p
Cut Cell DG	10^{-3}	1	73 000	18.95
	10^{-3}	2	72 000	17.18
	10^{-3}	3	70 500	17.00
Wan and Turek (2006)			139 000	17.42
			354 000	17.15
Glowinski et al. (2001)	7.5×10^{-4}		1 768 000	17.31

Table 5.6 shows results of the calculations. Here, a decrease of particle Reynolds number can be observed if the polynomial degree is increased. Thus, the drag force acting on the particle is underestimated with a low order approximation and the particle speed increases. The same observation has been made by Wan and Turek (2006). Additionally the results are compared to Glowinski et al. (2001), who use a

⁴Modified and extended version of (Krause and Kummer (2017), Section 4.4)

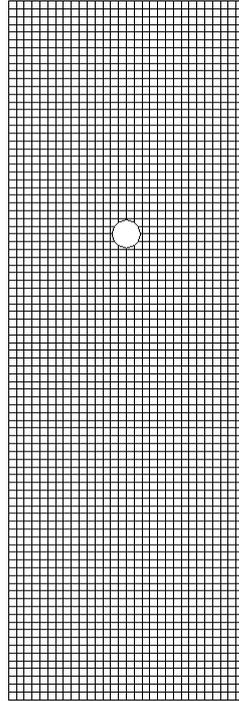


Figure 5.10: Background mesh at $t = t_0$ for calculation with polynomial degree of $k = 3$.

implicit immersed boundary approach with lagrange multipliers. In summary, using an explicit coupling approach yields a smaller particle Reynolds number than using implicit coupling. However, the total falling velocity appears to be in a similar range as the results from the references cited above. Again, better results can be obtained by increasing the polynomial degree and keeping the total DoF constant, which confirms the accuracy also for coupled motion.

In Figure 5.11 the particle position and the particle Reynolds number are plotted over time. The crosses indicate the termination time of the calculation when the particle comes too close to the lower wall and the simulation was stopped, because no collision model was implemented at that point. It can be observed, that the finer the background mesh the earlier the simulation has to stop. This is because the falling velocity of the particle at lower higher polynomial degrees decreases or in other words: The particle reaches the bottom of the domain faster, if a low polynomial degree is used.

For conducting a convergence study in time the timespan from release of the disk until $t = 0.02$ was used and divided by an increasing number of time steps, starting by choosing two. Also the same configuration as for the setting with a polynomial degree of three was being used. For each calculation the number of time steps were divided by 2 and a reference solution was calculated by using 256 time steps. The L^2 -Error for velocity can be seen in Figure 5.12.

As it is well known, because of the Lie-splitting approach, the current method renders to be only first order in time. Resulting out of this reason, the expected EOC can be confirmed by Figure 5.12.

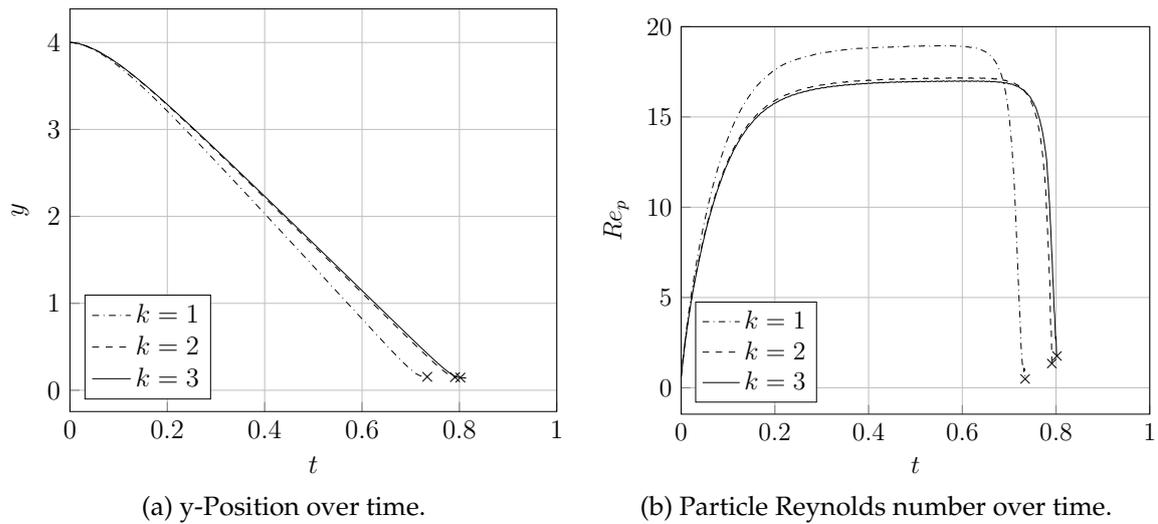


Figure 5.11: Results of a single disk falling in incompressible fluid.

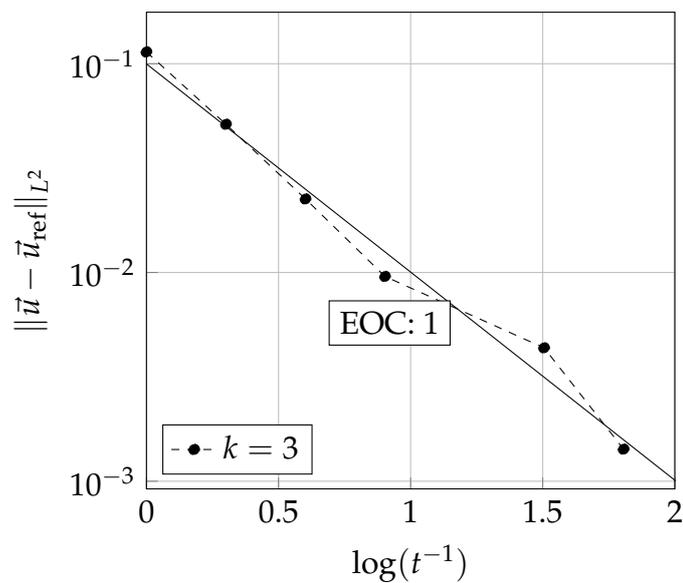


Figure 5.12: t-convergence of Lie-splitting.

Splitting vs. moving interface approach

However, an alternative approach which does not restrict the method to be first order in time will be considered in the following. Thus, this testcase is also considered to compare the splitting of our method with the moving interface approach by Kummer et al. (2017). In this approach the authors combine the advantages of splitting and space-time methods for time discretization. The main idea is to evaluate numerical fluxes on a moving interface frame by using a cell agglomeration strategy in time like it was shortly described in Chapter 4. Please note that this part extends the information which were given in the original publication of Krause and Kummer (2017).

As already mentioned, the splitting approach renders to be first order in time only. Therefore, the moving interface approach is a promising alternative if similar physical key values can be obtained in reasonable time.

At first, a refined setting is introduced rendering the total number of DoF to around 140 000. This means, if the polynomial degree is increased the grid is coarsened until almost the same DoF are reached. Results of those calculations using splitting and moving interface approaches using a fixed time step size can be seen in Table 5.7. For splitting, a high-order approximation in space leads to better convergence to the literature values whereas for the moving interface approach a convergence cannot be observed. For $k = 1$ the moving interface approach performs better than the splitting, possibly because it also takes effect during the interface movement into account. However, this cannot be confirmed for second and third order polynomials.

An explanation for this effect can be found in Kummer et al. (2017). Here, it is stated that evaluating a specific time-volume integral requires a polynomial degree for temporal discretization of order $2k$. As a result, for $k = 1$ a backward differencing formula (BDF)-scheme of at least second order is needed, which is obviously fulfilled by using backward differencing formula of second order (BDF-2). However, implicit methods like BDF are unstable for orders beyond 4 and thus the "temporal mesh" has to be refined by decreasing the time step size.

Table 5.7: Results splitting vs. moving interface in a refined setting.

Method	Δt	k	DoF	max Re_p
Cut Cell DG (Splitting)	10^{-3}	1	149 000	19.27
	10^{-3}	2	147 000	17.53
	10^{-3}	3	146 000	17.40
Cut Cell DG (Moving Interface)	10^{-3}	1	149 000	19.17
	10^{-3}	2	147 000	17.80
	10^{-3}	3	146 000	23.32
Wan and Turek (2006)			139 000	17.42
			354 000	17.15
Glowinski et al. (2001)	7.5×10^{-4}		1 768 000	17.31

In Table 5.8 this was done for the case of $k = 3$ and both approaches. The time step is decreased in two steps by a factor of 5 each time. Time convergence of the splitting technique is almost already reached for the time step size of $\Delta t = 10^{-3}$. In contrast, the time error for the moving interface approach is high for the largest time step. However, by decreasing the time step size calculations using the moving interface approach also converge to the same value for particle Re of $Re_p = 17.42$.

To conclude, a time discretization with the moving interface approach requires a fine time resolution to reach the same results.

Table 5.8: Dependency of moving domain approaches on time accuracy.

Method	Δt	k	DoF	max Re_p
Cut Cell DG (Splitting)	10^{-3}	3	146 000	17.40
	$5 \cdot 10^{-4}$	3	146 000	17.42
	10^{-4}	3	146 000	17.42
Cut Cell DG (Moving Interface)	10^{-3}	3	146 000	23.32
	$5 \cdot 10^{-4}$	3	146 000	17.72
	10^{-4}	3	146 000	17.42

5.1.5 Draft, kissing and tumbling of two circular particles

The complete model including collisions will be evaluated in a two-dimensional setting containing two particles falling in an incompressible fluid. The case has also been evaluated by Glowinski (2003) and Wan and Turek (2006) with their immersed boundary methods.

In Fortes et al. (1987) it was shown, that two disks dropped close to each other undergo draft, kissing and tumbling effects. First, the upper disk approaches the lower one because the effect of hydrodynamical force is significantly smaller for the upper disk (draft). Second, at a certain point in time they collide (kissing) and fall packed together until tumbling occurs, meaning the flow configuration of two objects falling in a row is unstable and they will separate from each other (tumble). Because of all these effects this test is well suited to evaluate the complete model.

The dimension of the computational domain is $\Omega = (-1, 1) \times (0, 8)$ and the radii of both particles are set to $R_0 = R_1 = 0.1$. The viscosity and density of the fluid are set to $\mu_f = 0.01$ and $\rho_f = 1.0$. The particles with density $\rho_{p1} = \rho_{p2} = 1.01$ are slightly heavier than the surrounding fluid. Initially the particles are at rest with $U_{p1} = U_{p2} = (0, 0)$ and $\omega_{p1} = \omega_{p2} = 0$ at positions $X_{p1} = (0, 7.2)$ and $X_{p2} = (0, 6.8)$. An adaptive refinement strategy is used to refine the mesh at the particle interface in order to resolve the hydrodynamical forces accurately. The walls are assumed to be fully plastic ($e = 0.0$) in order to prevent bouncing. Please note in the following the upper particle is denoted with particle 1 and the lower with particle 2.

An overview over the behavior of the two disks can be found in Figure 5.13. The velocity magnitude for different times is plotted to get an insight into this numerical test. In this figure, all important points in time during the calculation are plotted. At $t = 1.13$ the particles 'kiss' each other until they begin to tumble at $t = 1.74$. Subsequently, they separate at $t = 1.93$ leading particle 2 almost touching the left wall. Finally, at $t = 5.94$ and $t = 7.37$ both particles hit the bottom wall.

At first, the two collision models described in Section 4.3 will be compared to each other. In the following those two models are abbreviated as follows: (i) Momentum conservation model = Mom ; (ii) Repulsive force model = Rep, see Section 4.3. For comparison, a setting of 3600 cells (54 000 DoF) with a polynomial degree of $k = 2$

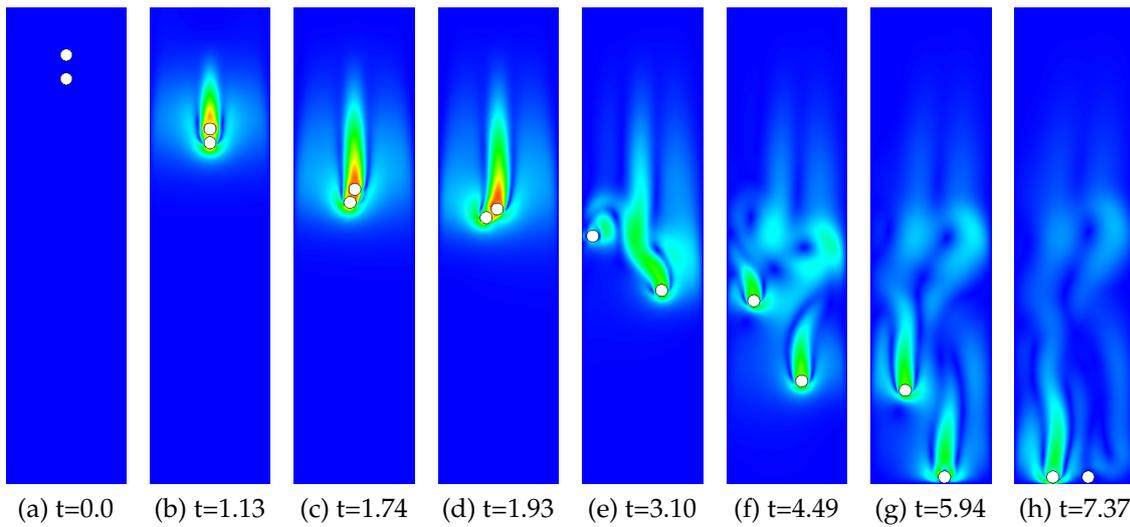


Figure 5.13: Velocity magnitude at different times t . The effects of draft, kissing and tumbling can be observed. Here the collision model based on the momentum exchange is used.

is chosen. Note, that an adaptive mesh refinement is also being used for refinement at the particle surfaces. In Figure 5.14 the positions of both particles over time are plotted. Each plot contains the position of particle 1 and particle 2 for both collision models. In the y -position plot the moment of collision with the bottom wall can be seen. By using the repulsive force model, particle 1 collides with the bottom at $t = 5.64$ whereas with the conservation of momentum model the collision takes place slightly later ($t = 5.78$). The same behavior can also be confirmed for the second particle which collides again first ($t = 6.94$) with the usage of the repulsive force model in contrast to $t = 7.00$. Moreover, it can be seen that the particle-particle collision with the momentum conservation model leads to an earlier acceleration in positive x -direction. However, both models are in very good agreement with the behavior described by Wan and Turek (2006) until the bottom wall is being hit. After the impact, the repulsive force model for particle-wall collision follows a different behavior including re-bouncing. Nevertheless, since both models are in good agreement the model based on momentum conservation will be used in all following calculations.

For comparing values with the work of Wan and Turek (2006), the mesh is further refined to 6400 cells (96 000 DoF). Again x - and y -position over time are plotted in Figure 5.15. The refined mesh is denoted with subscript 'h'. The general behavior of both calculations is the same: All effects in this test can be identified for both calculations. However, it can easily be seen that the refined mesh leads to a significant later occurrence of the tumbling effect. As a result both particles hit the bottom wall later, because they are not further accelerated until the tumbling occurs. The sensitiveness of small perturbations for the x -direction in which the tumbling takes place is confirmed by the plot of the x -position over time. Here tumbling occurs in negative x -direction for the refined mesh in contrast to the previous investigated resolution.

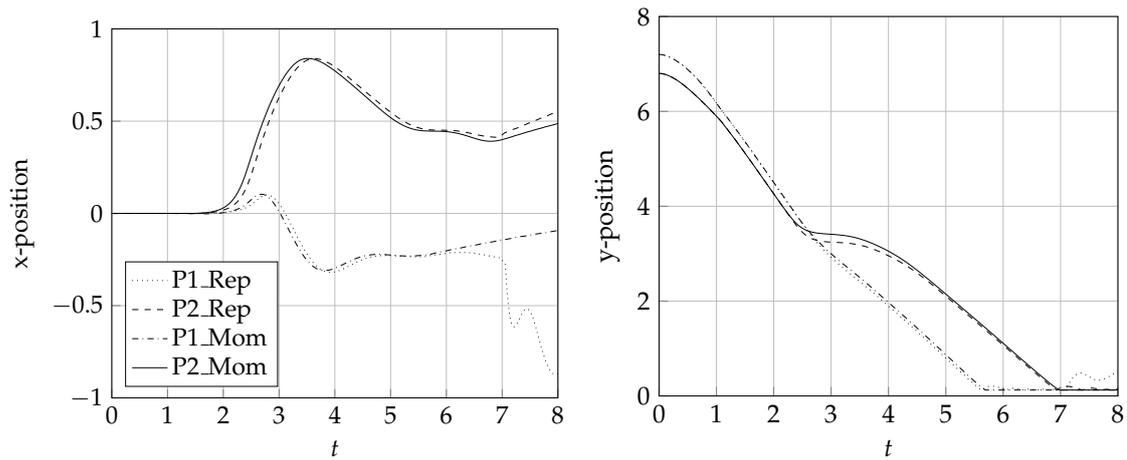


Figure 5.14: Comparison of collision models with $k = 2$ for the trajectories of both particles.

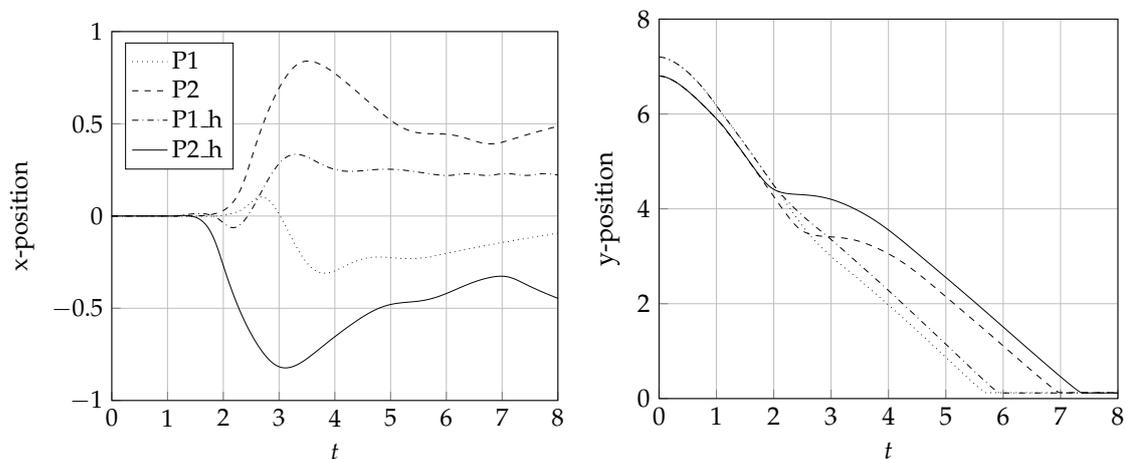


Figure 5.15: Behavior under h -refinement for conservation model with $k = 2$. The refined mesh is denoted with 'h' for both particles.

All physical effects of this test can be observed. In a next step, the results are compared with literature. In Table 5.9 the points in time where those effects occur are compared with those of Wan and Turek (2006) using a repulsive force collision model. There exist no exact definition for tumbling and separation whereas for particle-particle and particle-wall collisions, the exact time when the collision model is triggered can be used. Thus, for tumbling and separation the plots of the given literature are used to determine the time.

The 'kissing' is in very good agreement with the given literature, however the tumbling and separation occurs later ($\Delta t \approx 0.2$). This leads to the fact that all other effects during the calculation occur earlier than in comparison with the cited literature. However, since Wan and Turek (2006) also compare their results only qualitatively with other works it can be stated that the presented method leads to slightly different behavior. In a qualitative perspective it is in very good agreement.

Table 5.9: Points in time where collision effects occur for the calculation of the refined mesh in Figure 5.15.

Effect	Cut Cell DG	Wan and Turek (2006)
Kiss	1.13	1.13
Tumble	1.74	1.53
Separate	1.93	1.73
P1 hits bottom	5.94	6.23
P2 hits bottom	7.37	7.57

5.2 Particles with non-circular shape

Next, the presented solver is extended to particles with non-spherical shape. Meaning that for the calculation of hydrodynamical force and torque the distance between a quadrature node and the center of mass has to be accounted. In addition, the collision model based on momentum conservation is tested for eccentric collisions. Those lead to larger rotational effects after collision.

5.2.1 Elliptic disk falling with various initial angle

The solver will be evaluated for non-spherical shaped particles, e.g. an ellipse falling in an incompressible fluid. This test is a self-made benchmark for both, calculation of hydrodynamical forces and torque as well as evaluation of the particle-wall collision.

In detail, the calculation of hydrodynamical force and torques has to represent the real physical behavior of an ellipse. Therefore, a non-constant distance between the particle surface and the center of mass as well as rotation of the particle comes into play. For this, a channel has been set up with $\Omega = (-1.5, 1.5) \times (0, 4)$ where both, the fluid and the ellipse are initially at rest. The densities are $\rho_f = 1.0$ for the fluid and $\rho_p = 10.0$ for the ellipse to show the stability of the method for a higher density ratio. The viscosity is chosen to be $\mu = 0.1$. The gravity constant is chosen to be $g = -9.81$. Further, the shape parameters are chosen to be $a = 0.3$ and $b = 0.1$. The ellipse is placed at position $\vec{X}_p = (0, 3)$ and is initially at rest. The particle-wall collision is assumed to be inelastic with $e = 0.5$.

In the following calculations the polynomial degree k and the starting angle of the ellipse is varied. In Figure 5.16, the starting positions of all three different settings can be seen. All settings are presented in Table 5.10. It can be seen that for all settings an adaptive mesh refinement strategy is used to better resolve the ellipsoidal boundary for the wall collision which occurs during the simulation. This is easy to realize by using the DG method. An equidistant grid is used with 60 times 80 cells. Along the disk boundary the mesh is refined in two levels using a 2:1 ratio to the next level. The same mesh is used for all calculations, however the total number of DoF increase because of the different polynomial degrees used.

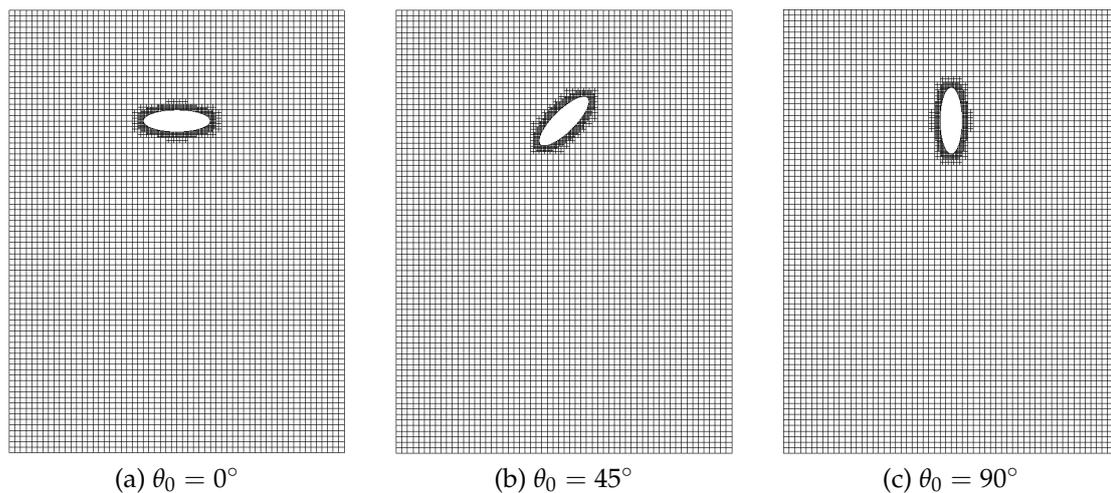


Figure 5.16: Different starting angles of an elliptic disk falling in incompressible fluid.

Table 5.10: Setting for falling elliptic disk.

k	DoF at t^0	Adaptive Mesh
1	33 600	yes
2	72 000	yes
3	124 800	yes

The values for x-position, y-position, x-velocity and y-velocity are plotted over time and compared for different polynomial degrees k , leading to increasing spatial resolution. All calculations will be also carried out after the ellipse collided with the bottom wall until a termination time $t_{end} = 2.0$ is reached. For all collisions the model based on the conservation of momentum is being used.

Elliptic disk with 0°

At first, the disk is placed with a starting angle of $\theta_p = 0^\circ$ in the channel. Of course, this starting angle is expected to result in the slowest possible falling velocity, not depending on spatial resolution. In Figure 5.17 the results can be seen. It is clearly notable that all quantities coincide perfectly for the free-fall part. In all calculations the ellipse is falling with the same speed and no distraction in x-direction. In detail, this means that the ellipse is not tilted during its falling process. Right before reaching the wall, the ellipse is slowing down due to the presence of the bottom-wall. The ellipse is slowed down due to hydrodynamical forces such that the collision model is first active at $t_{col} = 1.77$.

The behavior of the elliptic disk after collision differs slightly. For a polynomial degree of $k = 3$, a small velocity in x-direction occurs after the collision model is triggered. However, reaching and staying in a position of rest is also the most challenging part for a collision model. This is due to the equilibrium state between gravitational forces and collision forces acting from the wall unto the elliptic disk. As the sign of the

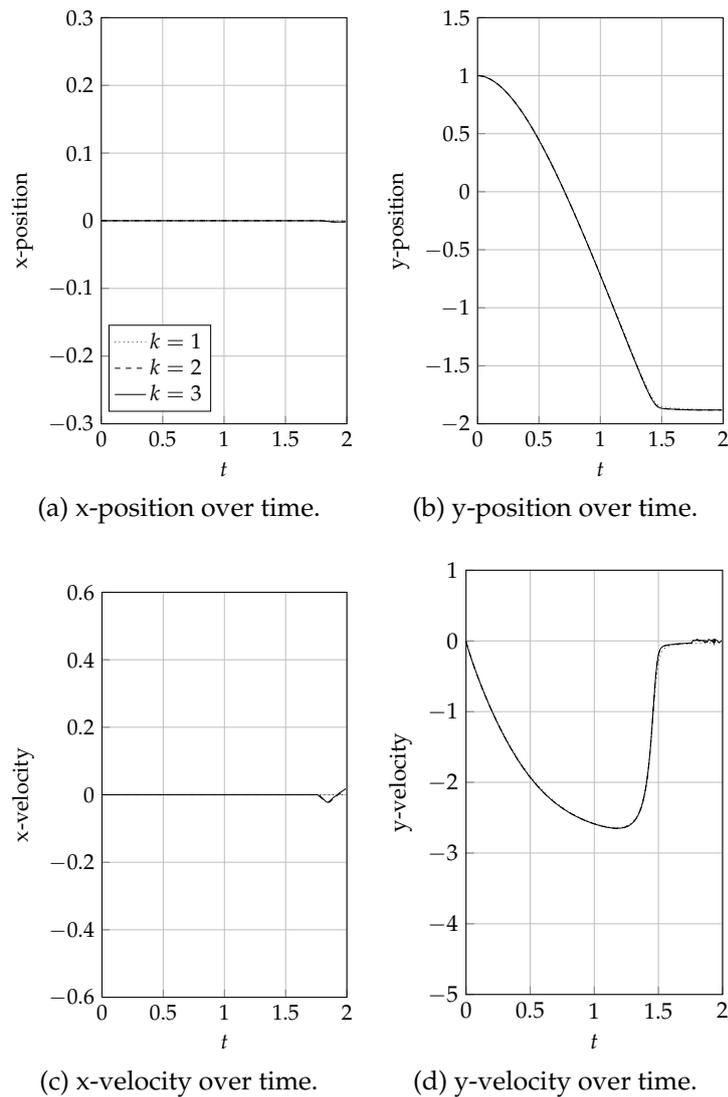


Figure 5.17: Results of an ellipse falling with $\theta_0 = 0^\circ$.

momentum is changed in every timestep, an "oscillatory" equilibrium state has to be kept by the collision model.

The comparison of all calculations leads to the conclusion, that those instabilities can be damped by choosing a low polynomial degree, leading to 'stable' position of rest. Therefore, for high-order calculations the presented collision model has to be improved. Nonetheless, a general prediction of the point of collision and the falling trajectories during the free fall process is not sensitive to spatial resolution.

Elliptic disk with 45°

In the next case, the elliptic disk is tilted 45° . The simulations are carried out with the same spatial resolutions. It can be predicted that a tilted disk will gain an acceleration in x-direction due to hydrodynamical forces. In Figure 5.18, it can be seen that for all

resolutions the acceleration in horizontal direction is present. In addition, the time of collision is identical for all polynomial degrees and can be denoted with $t_{\text{col}} = 1.14$.

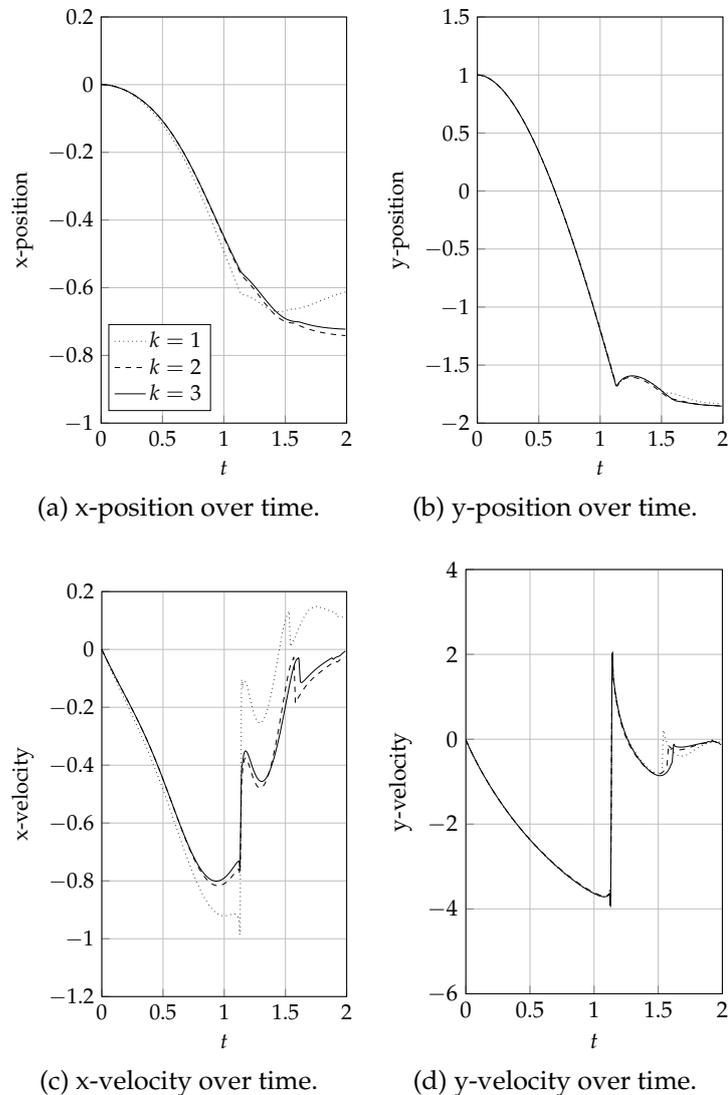


Figure 5.18: Results of an elliptic disk falling with $\theta_0 = 45^\circ$.

However, in contrast to the first setting with $\theta_0 = 0^\circ$, some differences between resolutions occur. First, it will be focused on the free falling part of the calculation for $t < t_{\text{col}}$. The vertical positions and velocities coincide quite well between all simulations. For the x-Position it can be seen, that for the coarsest spatial resolution an overprediction in hydrodynamical force takes place. Therefore, the ellipse is accelerated more in x-direction than for $k = 2$ and $k = 3$.

Lastly, the x-position and velocity differs much depending on resolution. For $k = 1$ the x-Position at $t = 2.0$ renders to be close to $x = -0.6$ whereas for both other calculations the position of rest is at $x = -0.71$. The aforementioned overprediction of x-Velocity leads to a different behavior after collision as well. Here, the convergence of $k = 2$ and

$k = 3$ to a position of rest can be seen, whereas the calculation for $k = 1$ still moves in positive x-direction.

Elliptic disk with 90°

In the last variation of the current problem, the elliptic disk is tilted by 90° . This leads to the fastest falling velocity. Therefore, a collision time which is smaller than in the calculations with angles of 0° and 45° can be expected. Results of this calculation can be seen in Figure 5.19. Here, the collision time can be denoted with $t_{\text{col}} = 1.06$ for a first collision and with $t_{2\text{ndcol}} = 1.55$ for a second collision. Nonetheless, until the time of the first collision large agreement for the free-falling part can be yield. Only the velocity in vertical direction differs slightly at $k = 1$ but has almost no impact on the collision time.

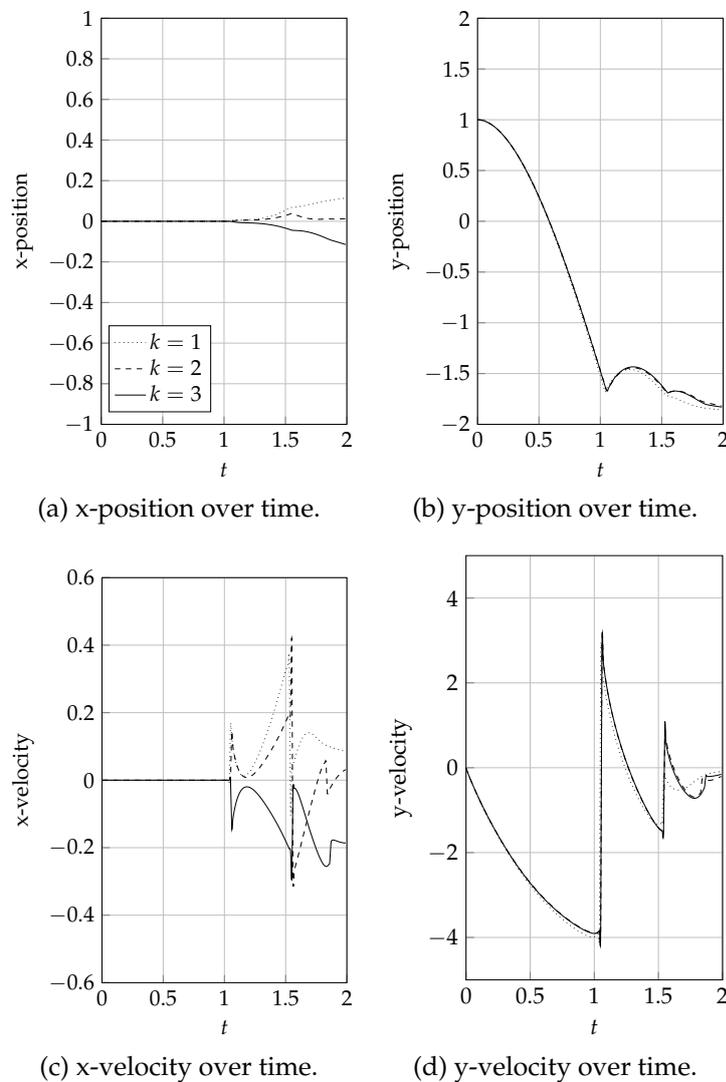


Figure 5.19: Results of an elliptic disk with $\theta_0 = 90^\circ$.

After colliding the elliptic disk increase velocity in x-direction for all three cases. This results from a normal collision vector which is not exactly $\vec{n} = \{0, 1\}$. Resulting out

of this, the acceleration in the opposite x-direction for $k = 3$ occurs. However, if the x-position and x-velocity is mirrored at $x = 0$, resp. $U_x = 0$, almost no difference between $k = 2$ and $k = 3$ can be seen. Therefore, a convergence can be observed.

For $t > t_{2\text{ndcol}}$, the most significant difference is the already mentioned velocity increase after the second collision. For $k = 1$ and $k = 3$ the velocity at $t = 2.0$ is far from rest. This means, that the ellipse is still moving in x-direction whereas in y-direction it is almost at rest. This happens due to different turns in orientation leading to increasing hydrodynamical forces in horizontal direction. However, convergence in y-position and y-direction can be seen for all investigated cases.

All in all, the current test shows the ability of the method to predict the falling behavior even for coarse sparse resolutions. However, as soon as strong antisymmetric hydrodynamical effects and collisions come into play, a fine resolution at the interface for integration and collision determination is required. It also shows the big challenge of reaching convergence for a numerical scheme of collisions which is very sensitive to fluctuations in hydrodynamical forces and collision direction calculation.

5.2.2 Dry collisions of multiple particles

In this test, the cut cell collision model based on conservation of momentum presented in Section 4.3 is evaluated for different particle shapes. The current shapes are extended to a squircle, a bean and a hippopede. Those three different shapes also have concave boundaries and thus collision detection and closest point finding are more complicated. Additionally, this is the first numerical test where various eccentric collisions come into play.

Please note, that this test is performed in a 'dry' environment, which means only rigid body motion is applied without fluid. Leading to only solving NEE. Of course, this leads to a simple test of collision detection and collision model without incorporating the complete two-way coupling.

Table 5.11: Initial particle settings for collision test.

Shape	\vec{X}_0	θ_0	\vec{U}_0	ω_0
Disk	$\{-0.6, 0.3\}$	-90°	$\{0, 0\}$	0
Ellipse	$\{-1.2, 0.9\}$	90°	$\{-5, 0\}$	-10
Squircle	$\{-1.0, 1.0\}$	-20°	$\{-5, -5\}$	0
Bean	$\{-1.0, -1.0\}$	-20°	$\{-5, 5\}$	-10
Hippopede	$\{-0.21, -0.5\}$	-45°	$\{-5, 0\}$	0

In Table 5.11, the starting positions and velocities for all particles can be seen. This table is useful for understanding the sequence presented in Figure 5.20. Different starting angles and starting velocities have been used in order to apply an initial momentum to the whole system. During the progress of the calculation various different collisions of particles and particles with walls occur. This is important because in case of an

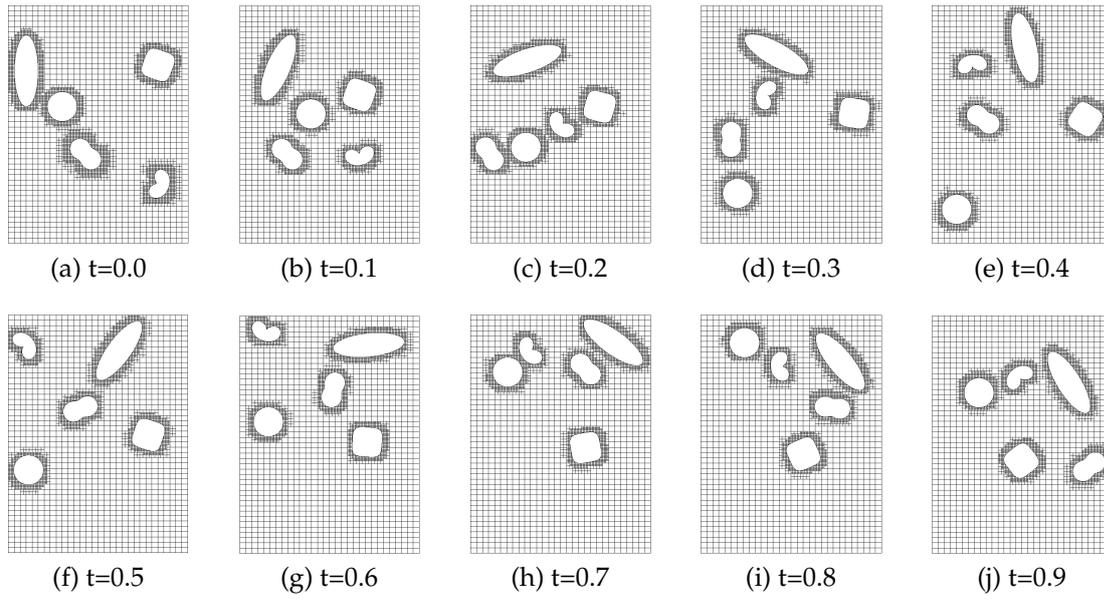


Figure 5.20: View on the collision test at different points in time. Collisions take place in a 'dry' environment, meaning no fluid flow is present.

overlap, the simulation will break down due to the representation of the particles using a characteristic function.

Besides testing the particle-particle and particle-wall collision model for robustness, total momentum and kinetic energy in the system are also tracked. The kinetic energy is determined throughout the computation using (5.7). As already mentioned, momentum conservation is only maintained during particle-particle collisions whereas conservation of kinetic energy is fulfilled for every collision scenario.

$$E_{\text{kin}} = \frac{1}{2} M_j |\vec{U}_j|^2 + \frac{1}{2} I_j \omega_j^2 \quad (5.7)$$

This test shows the method to be robust for several possible collision configurations. Further, the total kinetic energy is constant during this test. Of course, the collision model is conservative by construction but this shows its correct implementation. Note that exact trajectories of those particles will not be plotted because it is not suitable to have benchmark character and is only presented through reasons of completeness in this thesis.

5.2.3 Five particles of different shape falling in fluid

This test is chosen to show the ability of the overall method coupling all presented features together. The numerical result is considered as a proof of concept for the presented work to simulate particles with various shapes, two-way coupling and particle-particle as well as particle-wall collisions.

Here, all collisions are assumed to be fully elastic ($e = 1.0$). The density of all particles is denoted with $\rho_p = 3.0$ whereas the density of the fluid is $\rho_f = 1.0$. The particles are accelerated due to gravity with $g = -98.0$. The starting positions and angles can be seen in Table 5.12. Thus, all particles are placed randomly in the very upper part of the vertical channel. For this calculation a mesh containing 1800 cells (12600 DoF) is used together with an adaptive mesh refinement strategy at the particle surfaces. The polynomial degree of $k = 1$ is used for velocity. As already mentioned this is a proof of concept with a stable setting. For high order settings with different shaped particles an adaptive time step strategy is needed if particles come to close.

Table 5.12: Initial particle settings for five particles falling.

Shape	\vec{X}_0	θ_0
Disk	$\{-0.2, 7.5\}$	0°
large Disk	$\{-0.5, 7.2\}$	0°
Ellipse	$\{0.2, 7.3\}$	30°
Squiracle	$\{-0.2, 6.95\}$	-20°
large Squiracle	$\{0.2, 6.5\}$	-45°

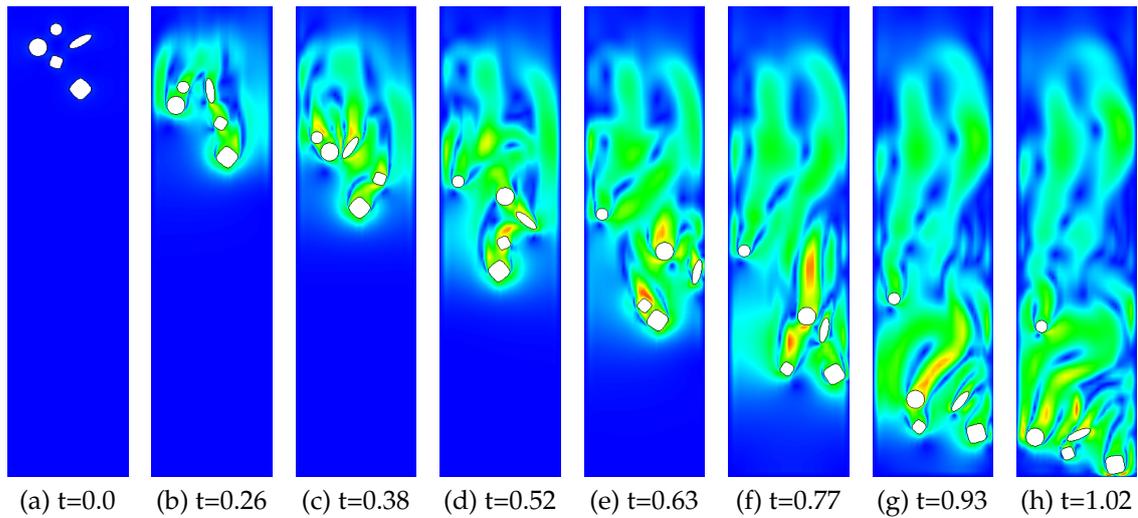


Figure 5.21: Velocity magnitude of particles with different shapes falling in incompressible fluid.

In Figure 5.21, the positions of all particles can be seen at different time. In addition, the overall velocity magnitude is indicated by coloring. All particles accelerate due to gravity and several collisions take place. Clearly notable is the collision of the ellipse with the large disk right before $t = 0.38$ and the collision of the ellipse with the right wall before $t = 0.63$. In addition, the draft, kissing and tumbling effect can be observed for both squiracles ($0.52 < t < 0.77$). In the end, the large squiracle collides with the bottom wall at $t = 1.02$. Here, a significant increase in velocity magnitude is clearly notable because of the flow is accelerated in the small gap between particle surface and wall.

5.3 Extension to three dimensions

Lastly, the method is extended to three dimensional calculations. However, only immersed boundary settings for sphere flows were taken into account to show the general ability of the main part (cut cell DG) to work for three dimensional calculations as well.

5.3.1 Stationary flow around a sphere

In real applications three dimensional particulate flows are required. In order to evaluate the cut cell DG method for three dimensional flows, a simple flow around a stationary sphere is considered.

The dimensions of the domain are $\Omega = [-10, 30] \times [-10, 10] \times [-10, 10]$. A sphere with radius $R = 5$ is immersed in the domain at center position of $[0,0,0]$. The boundary and initial conditions for this testcase can be extracted from (5.8) already for a time dependent problem. Hanging nodes are used to refine the grid at the sphere surface such that in total 400 000 DoF have to be solved in the coupled equation system.

$$\begin{cases} \vec{u}_D = 1 & \text{on } \Gamma_D = \{(x, y, z, t) \in \mathbb{R}^4; x = -10\} \\ p_D = 0 & \text{on } \partial\Omega \setminus \Gamma_D = \{(x, y, z) \in \mathbb{R}^3\} \\ \vec{u}_0 = \{1, 0, 0\} & \text{in } \Omega = (-10, 30) \times (-10, 10) \times (-10, 10) \text{ for } t = t_0 \end{cases} \quad (5.8)$$

The flow is evaluated at $Re=100$, which renders the flow field to be stationary. The results of the evaluated drag coefficient can be seen in Table 5.13. The accuracy is increased with polynomial degree and is closer to the immersed boundary results of Mittal (1999) than to those of Fadlun et al. (2000).

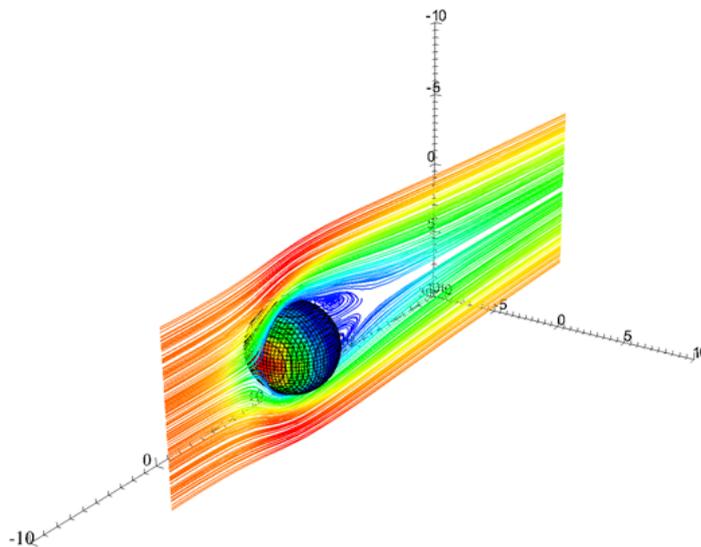


Figure 5.22: Velocity magnitude and streamlines of a sphere flow with $Re=100$.

Table 5.13: Results of a sphere flow at $Re=100$.

Method	k	DoF	C_D
Cut Cell DG	1	400 000	1.0004
	2	400 000	1.1051
Fadlun et al. (2000)			1.0794
Mittal (1999)			1.0900

Although the blocking ratio between the sphere and the domain boundary is not optimal, meaning the possible influence of the boundary condition on the physical quantities because of small distance, the results obtained are in good agreement. Therefore, this also renders the method to be of high potential in three dimensions.

5.3.2 Transient flow around a sphere

As the aforementioned calculation was carried out in a stationary setting, a transient sphere flow at $Re = 700$ is investigated in the following. Here, the simulation is carried out in the same setting but a timestep size of $\Delta t = 0.1$ is being chosen. The temporal accuracy of our timestepping scheme and the choice of the timestep also play an important role.

Table 5.14: Results of a sphere flow at $Re=700$.

Method	Δt	k	DoF	$\overline{C_D}$
Cut Cell DG	0.1	2	400 000	0.5445
Campregher et al. (2009)				0.5050
Morsi and Alexander (1972)				0.5000

In Table 5.14 it can be seen that the mean drag difference between our method and the literature has become larger. As a reason the spatial resolution of the mesh is too coarse to get valuable results, specially at the boundary layer. Unfortunately this is due to computational efficiency of the method. Nevertheless the key point to draw here is that time dependent three dimensional calculations are possible in general by applying the proposed method.

5.4 Conclusion

This chapter contains various numerical tests for validating the method introduced in Chapter 4. The presented computations are built on increasing complexity including more and more difficulties like coupling, motion, collisions or non-spherical particle shapes. Those challenges are always validated separately until a more advanced test couples two features together. This procedure is followed throughout this chapter.

First, the IBM method is tested with simple cylinder flows. Then, the two-way coupling using the calculation of hydrodynamic forces and the imposition of boundary conditions at the particle surfaces are validated using a rotating particle in shear flow. Here, the high-order spatial convergence of $\mathcal{O}(h^{k+1})$ is also shown. For moving domains the whole method renders to be of $\mathcal{O}(\Delta t + h^{k+1})$ because of Lie-splitting.

Those computations are followed by investigating draft, kissing and tumbling effects of two interacting particles in a flow field. This test is also used to compare the collision models based on repulsive force and momentum conservation. Both results are almost the same, such that for further calculations the momentum conservation model is used. Qualitative good agreement with the work of Wan and Turek (2006) is obtained as well.

The calculation of hydrodynamic forces and torque as well as eccentric collisions are validated considering a falling ellipse with different angle and a 'dry' collision of particles with various non-spherical shapes. In a next step, all parts are coupled together such that the ability of the solver to compute interactions between particles of different shape is proven.

In the end, the method is validated for the three dimensional calculation of steady and transient sphere flow with only partly satisfying results because of the coarse mesh size which had to be used. Due to the increase in DoF and computationally expensive quadrature in three dimensions this finally leads to the following chapter. Here, the focus is laid on introducing a Newton-Krylov iterative solver. The work presented in the next chapter further aims to increase computational efficiency mainly through parallelization.

6 Performance analysis and profiling

For every computational code in industry or academia simulating physical behavior the following questions can be stated:

1. Is the most efficient algorithm for solving the algebraic equations being used?
2. How can the performance of the method be measured?
3. How can performance issues be tackled?

This chapter aims to answer those questions for the method proposed in Chapter 4.

The chapter starts with a state of the art section, containing a literature review on the solution of equation systems and basics about performance measuring in the HPC context. Further, linearization techniques for the nonlinear equation systems are introduced and the importance of preconditioning is stated. Afterwards, different preconditioner will be tested for Picard and Newton linearized systems solved with the generalized minimal residual method (GMRES) method. In the last part of the chapter, a workflow for performance analysis of the *BoSSS*-code is proposed together with performance measurements and tuning results. Moreover, the current key bottlenecks of the underlying cut cell DG solver are worked out. As usual, this chapter ends with a conclusion.

6.1 State of the art

The analysis of performance is important for software development, especially if it comes to programs describing physical problems with high complexity. Many physical models are described by partial differential equation (PDE)s like the NSE for fluid flow, which have to be discretized by a numerical scheme, assembling a large equation system. CFD is one of the largest engineering application for HPC and its usage is under steady growth in industry and academia (Cant, 2002; Jayanti, 2018).

In engineering, a typical problem has a very large range in time and length scales which have to be resolved. For the particulate flow applications in this thesis, the accurate resolution of flow around particle surfaces in terms of collisions is a good example for small length-scales. Moreover, small time-scales have to be resolved for an accurate modeling of collision effects and collision time. This is especially the case for direct numerical simulation (DNS), in which all features are resolved and thus no modeling is required. For DNS in context of particulate flows, e.g. the work of Chouippe and Uhlmann (2015) can be mentioned. They consider spherical particles in a turbulent background flow.

In contrast to the aforementioned work, the method in this thesis aims to resolve arbitrary shaped particles in a laminar flow context. In Chapter 5 it is shown, that the DG method saves DoF by using high-order polynomials for the velocity and pressure space. Therefore, this advantage renders DG to be very suitable for DNS. Furthermore, DG has very good properties in terms of parallelization due to its cell local formulation and the coupling only between neighboring cells. At last, adaptivity (h -adaptivity) of the mesh and the polynomial degree (k -adaptivity) is possible.

6.1.1 Equation system

Usually, solving the equation system which is built by a discretization scheme is the largest bottleneck of a computation. Thus, increasing the speed for solving the equation system increases the whole computational performance significantly. Therefore, efficient algorithms for solving large equation systems have been a very active field of research until today. Depending on the method of choice for discretization, the system matrix may look different and certain methods are more applicable than others. In the following the fully coupled DG discretization matrix is considered, meaning momentum and continuity equation of the NSE are solved together. Further, it is focused on how to solve the system both, efficient and fast.

In the past, high-order DG methods have been applied for fluid flow problems, solving both, the incompressible (Bassi et al., 2007; Shahbazi et al., 2007; Klein et al., 2012) and the compressible (Ferrari et al., 2010; Fechter and Munz, 2015; Müller et al., 2016) NSE as well as for turbulent flows, e.g. Crivellini et al. (2013); Gassner and Beck (2013).

It has been noted, if the computational domain is large and three dimensional, those equation systems becomes difficult to solve by a direct solver based on Gaussian elimination because of memory consumption due to the storage of factorization. This leads to the question, which iterative solver can be used to solve the DG system accurately and also speeds up properly for parallel computations. Therefore, several methods for treating the nonlinearity of the Navier-Stokes equations are tested in combination with the GMRES algorithm proposed by (Saad and Schultz, 1986). This leads to a standard Picard fixpoint iteration in combination with GMRES and a so called Newton-Krylov method using the GMRES method to approximate the directional inverse of the Jacobian.

A big advantage of the GMRES method is the matrix-free implementation, which benefits in saving memory and is crucial for large computations. Nevertheless, GMRES has to be preconditioned to reach a proper convergence rate. Efficient and effective preconditioning is still under active research, see e.g. Kay et al. (2002); Silvester et al. (2001); Elman et al. (2002); Antonietti et al. (2017); Franciolini et al. (2017).

6.1.2 High-performance computing

As already mentioned before, complex CFD simulations need lots of computing power and memory in order to obtain a result in a reasonable time. There exist no exact definition for HPC, nevertheless it can be stated that problems which cannot be solved

on local workstations are tasks of HPC. Further, HPC is based on super-computing clusters which essentially gain performance through the execution of parallel tasks on several cores and large memory data drives. This section is mainly based on Rabenseifner (2015), further information can be found in the reference cited.

Of course in context of HPC, the law of Moore (1965) has to be mentioned. Moore (1965) stated, that the number of transistors in a dense integrated circuit doubles about every two years. Essentially, this leads to exponential growing compute power. Since in the past computing power was always cheaper than manpower this fact led to a lack of program optimization (Osterhage, 2016). Today, stagnation of computing power is predicted by some computer scientists for future years. Therefore, efficient parallel programming techniques and research in HPC becomes increasingly important.

Since HPC is about gaining performance using parallel computing architectures, some definitions have to be introduced:

- Let $T(p, N)$ be the time to solve a problem with N DoF on p processors.
- Parallel speedup is denoted with $S(p, N) = T(1, N) / T(p, N)$, optimally computing the same problem with more processors in less time (strong scaling).
- Let $T(p, p \cdot n) / T(1, n)$ be a larger problem with growing p , such that the size per process is fixed (weak scaling).

The parallel speedup is aimed to be equal to p in an optimal case, i.e. $S \sim p$. However, this can only be achieved in theory, reasoned by Amadahl's law (Amdahl, 1967), which will be introduced in the following: In the work of Amdahl (1967), a programs total runtime T has to be split in time which is serial t_S plus time which can be proceeded in parallel t_P . Then, the overall speedup can be estimated following (6.1):

$$S_A = \frac{T(p, N)}{t_S + \frac{t_P}{p}}. \quad (6.1)$$

In words, the theoretical speedup is always bound by the part of the task which cannot be parallelized. Amadahl's law is a pessimistic assessment of parallel performance. Therefore, Gustafson (1988) proposed a reevaluation of Amadahl's law because he showed, that it is too pessimistic in terms of massive ensemble parallelism. One can imagine that for larger problems in CFD applications, the part which can be proceeded in parallel grows much faster, whereas the sequential part remains almost constant. Gustafson (1988) proposed a linear approach of the form

$$S_G = \frac{t_S}{T(p, N)} + p \cdot \frac{t_P}{T(p, N)}. \quad (6.2)$$

There are numerous alternative proposed laws for parallel scaling behavior. For a another important law, which improves the both aforementioned ones, see e.g. Sun and Ni (1993). However, this goes beyond the scope of this thesis.

Parallel hardware architectures

For understanding the basics of parallel computing, at first, the most common hardware architectures have to be introduced. Those architectures are present in most HPC systems and are suitable for different aspects of parallel computing. In the following, three main types of parallel systems will be introduced:

- Shared memory systems
- Distributed memory systems
- Hybrid systems

However, variations of parallelization architectures and modifications like cache-only memory architecture (COMA) exist, see Dahlgren and Torrellas (1999).

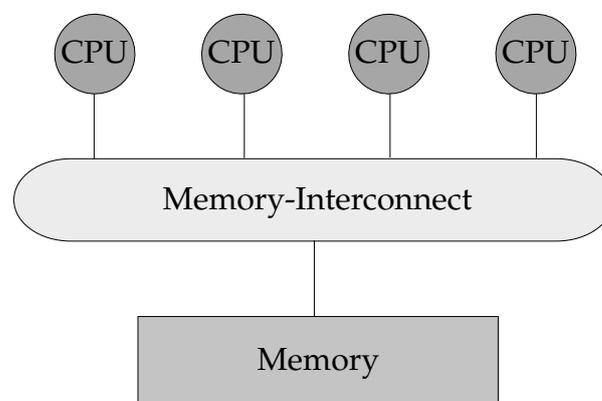


Figure 6.1: Shared memory system, in which each CPU can contain several cores.

The first architecture to mention is a shared memory system, which can be seen in Figure 6.1. In those systems all cores are connected to the memory bank via memory-interconnect, which has the same access speed for every core. Because of their uniform symmetric pattern, they are also called uniform memory access (UMA) or symmetric multi-processing (SMP) nodes.

In Figure 6.2, a distributed memory system can be seen. Here, the memory is only denoted with 'M'. In this case, all CPUs have fast access to their own memory but slower access to memory of all other CPUs. Various network types can be used for the node interconnect. The most commonly used network standard used in HPC is InfiniBand, e.g. see Buyya et al. (2002). Related to the different memory access speeds, those nodes are named non-uniform memory access (NUMA).

Modern computing cluster architectures are a combination of the two aforementioned ones. Typically, they are clusters of SMP nodes with fast access to their own memory and a node-interconnect network between them. A schematic sketch can be seen in Figure 6.3. Hybrid systems allow large computations to run thousands of cores in parallel and are optimized for fast access of SMP node memory.

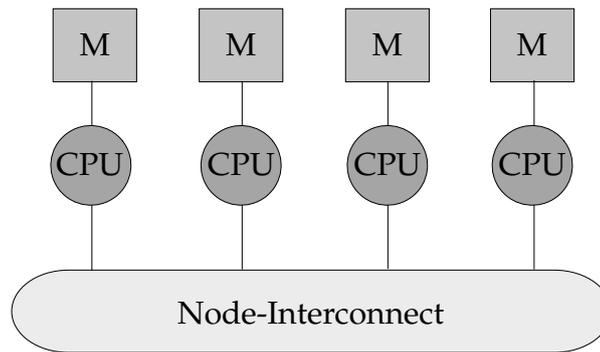


Figure 6.2: Distributed memory system with memory connected to each CPU.

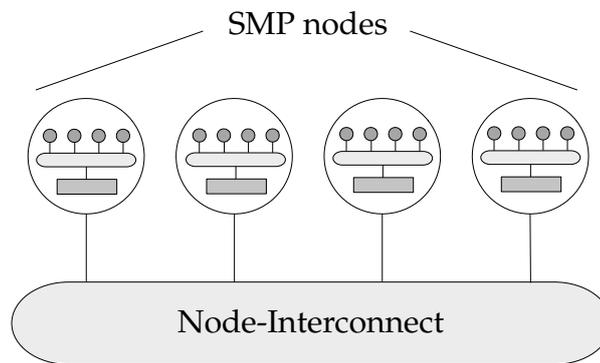


Figure 6.3: Hybrid memory system containing SMP nodes connected via network.

Parallel programming models

In the following, basic programming models for all mentioned hardware architectures in the previous section will be introduced. In order to build the bridge between hardware architecture and programming models, a few words about distribution of computational load will be given: The two main resources of parallel computing clusters are processors and memory. For parallelization, work can be distributed to processors as well as data. An example is the distribution of parts of a loop to several processors (work) and information of certain parts of the domain (data). Synchronization of the distributed work and communication of data between processors has to be done during runtime. In addition to that, distribution of work and data in CFD can be done directly in the computational domain using domain decomposition. Those techniques are for example used for preconditioning, as it is described in Section 6.2.

In terms of shared memory systems, the Open multi-processing (OpenMP) application programming interface (API) is used the most (OpenMP Board, 2015). The programming model is thread-based and standardized since 1997. The user has to specify work decomposition, no data decomposition is needed due to shared memory access. The synchronization of work is usually implicit and not defined by the user. The main program parts which are parallelized are loops. Because of its requirement of shared memory architecture, its only feasible for a medium number of processors.

For distributed memory systems, another standardized approach is used, called message passing interface (MPI) (MPI Forum, 2015). In MPI, the user has to specify work and data distribution as well as communication between all processors. Synchronization is implicitly done after the completion of communication. For this, the application calls MPI library routines. The main advantage of MPI is, that it scales up to nearly any system size and can also be used on shared memory systems.

Certainly, OpenMP and MPI techniques can be mixed for hybrid systems, which is beyond the scope of this thesis. For information on hybrid parallel programming, see Rabenseifner and Wellein (2005). For general information to all topics in this section the reader is referred to the book of Grama et al. (2003) and the work of Rabenseifner (2015).

6.1.3 Software performance testing

Accurate and efficient measurement of performance is the key to optimize the source code. Measurements should be as detailed as possible but in addition, they preferably have little impact on the program behavior.

Firstly, it has to be defined which metric or quantity of a computation has to be measured. Possible metrics are call counts, time, memory consumption, network traffic, I/O or CPU power, see Osterhage (2016).

Benchmarks

It is important to previously answer the question if performance has to be measured for the overall code or a specific part, e.g. solving the equation system, quadrature or creating the mesh. With this a suitable benchmark can be assembled to tailor the needs of the user.

In general, benchmarks have to be reproducible in order to be a 'good' benchmark. Depending on what metric to measure, details of the benchmark environment like hardware, software, and configurations have to be recorded. In addition, enough repetitions of a benchmark lead to a more accurate value in order to decrease fluctuations due to network traffic or utilization of computing resources by other applications. With this, e.g. factors of network traffic can be determined, especially if the network is not used exclusively by the program to benchmark. Those factors can easily be subtracted from measurement times after benchmarking. As already mentioned, for benchmarking the machine has to be used exclusively. Benchmarks should be also automated as much as possible to protect from handmade mistakes made by the user.

Now, the differences between low-level and application benchmarks will be explained (Hager and Wellein, 2011). In low-level benchmarking, only a particular part of the code will be measured to evaluate, e.g. processor performance. If the timespan is very small it is useful to use high-precision timers for benchmarking. In contrast, application benchmarks aim in predicting the 'real' behavior of actual code runs. Presently, realistic applications like test cases in the context of CFD can be used. Application benchmarking is also the method of choice for checking if the right algorithm for a

certain task has been used or an architecture is more feasible than another one. Lastly, there exist parallel benchmarks especially for HPC system which run highly parallel. Here, the focus is mainly laid on communication time and synchronization. More information on benchmarking, can be extracted from Hager and Wellein (2011).

Profiling

For measuring performance quantities, there mainly exist two approaches: code instrumentation and sampling. Both have their advantages and disadvantages depending on the need of the user.

By using code instrumentation the compiler inserts calls to the code that measure time and also saves the complete call stack. Usually process or thread number is also recorded, which is useful for parallel performance analysis. This technique can cause massive overhead due to the code sections inserted, but generates definite counts on how often a method was called. Thus, it is more suitable for programs with short runtime.

In contrast, code sampling is less intrusive. Here, the code is interrupted and the program counter is recorded. As a result, sampling generates only statistical information on how often a function was found to be active, nevertheless its accuracy grows the longer the run of the code. It also can be applied with very little overhead and does not interfere with the compiler.

The output of both techniques are files, trace files for instrumentation, and profiling summaries for sampling. Those have to be analyzed by using software to visualize information. Of course, there is also software which directly analyzes the output, however, usually the analysis has to be done by the user in a human readable context.

The basic information contained in this part have been extracted from Hager and Wellein (2011) and Rabenseifner (2015). For further information please see these references.

6.2 Solving the coupled nonlinear equation system

As already mentioned in the beginning of this chapter, solving the equation system is the largest bottleneck of CFD. Therefore, in this section different solution strategies will be explained in detail.

The discretization of (2.5) according to Section 4.2 yields the system of equations

$$\hat{M} \frac{3\bar{x}^{n+1} - 2\bar{x}^n + \bar{x}^{n-1}}{\Delta t} + A(\bar{x}^n)\bar{x}^{n+1} = \vec{b}. \quad (6.3)$$

where $\hat{M} = \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{(m+n) \times (m+n)}$ denotes the modified mass matrix, resulting out of orthonormal basis functions without any cut cells. If cut cells are present, the

presented structure of the identity does not hold. For detailed information the reader is referred to Kummer et al. (2017).

Further, with $A(\vec{x}) = \begin{pmatrix} F(\vec{u}) & B^T \\ B & 0 \end{pmatrix} \in \mathbb{R}^{(m+n) \times (m+n)}$, $\vec{x} = \begin{pmatrix} \vec{u} \\ p \end{pmatrix} \in \mathbb{R}^{m+n}$, and $\vec{b} = \begin{pmatrix} \vec{f} \\ g \end{pmatrix} \in \mathbb{R}^{m+n}$, a so called saddle point system of (4.1) is obtained

$$\hat{M} \frac{3\vec{u}^{n+1} - 2\vec{u}^n + \vec{u}^{n-1}}{\Delta t} + \begin{pmatrix} F(\vec{u}) & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \vec{u} \\ p \end{pmatrix} = \begin{pmatrix} \vec{f} \\ g \end{pmatrix}. \quad (6.4)$$

In (6.4), $F(\vec{u})$ denotes the discretization matrix of the convection and diffusion operator, B^T the matrix of the pressure gradient discretization, which is the transposed of B , the one of the velocity divergence in discrete form. Note that $F(\vec{u})$ is non-symmetric for NSE. In this section, the mass matrix \hat{M} and terms of temporal derivatives will be dropped and a steady state solution will be considered since all solution methods for equation systems have to be applied in every timestep.

6.2.1 Nonlinear solver

The saddle point system (6.4) is nonlinear. This is due to the submatrix F depending on the current velocity values. For treating the nonlinearity there are two main approaches which will be pointed out in the following:

The first one is the so called Picard iteration approach. Hereby, the matrix F gets linearized by using the velocity of the previous iteration \vec{u}^n . Note that here the superscript n indicates nonlinear iterations. Therefore, in every nonlinear iteration a linear Oseen-system has to be solved by a linear solver, e.g GMRES:

$$\begin{pmatrix} F(\vec{u}^n) & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \vec{u}^{n+1} \\ p \end{pmatrix} = \begin{pmatrix} \vec{f} \\ g \end{pmatrix}. \quad (6.5)$$

Note, that the Picard iteration method has good convergence properties for sufficient small values of Δt . However, by using this method the solution of the nonlinear equation converges only linearly.

A method which is quadratically convergent is Newton's method. In order to formulate the Newton iteration step for the solution of $A(\vec{x})\vec{x} = \vec{b}$, the residuum system has to be defined

$$\vec{r}(\vec{x}^n) = A(\vec{x}^n)\vec{x}^n - \vec{b}. \quad (6.6)$$

With this, the Newton step can be stated as follows

$$\vec{x}^{n+1} = \vec{x}^n - (\vec{r}'(\vec{x}^n))^{-1}\vec{r}(\vec{x}^n) = \vec{x}^n + \Delta\vec{x}^n, \quad (6.7)$$

where $J(\vec{x}^n) = \vec{r}'(\vec{x}^n) = A'(\vec{x}^n)\vec{x}^n + A(\vec{x}^n)$ denotes the Jacobian matrix of the system matrix $A(\vec{x}^n)$, which is usually approximated by a finite difference ansatz in each coordinate direction.

Assembling and inverting the Jacobian is very time consuming. Therefore, instead of solving the system (6.7) directly, one solves the linear system

$$J(\vec{x}^n)\Delta\vec{x}^n = -\vec{r}(\vec{x}^n) \quad (6.8)$$

by using a matrix-free approach, e.g. a Krylov-subspace method which will be explained further in Subsection 6.2.2.

6.2.2 Linear solver

If the system matrix is small enough, the linearized system matrix can be solved by direct solver libraries like PARDISO (Schenk and Gärtner, 2002, 2004, 2006) or MUMPS (Amestoy et al., 2000). Note that those solvers are easy to apply to system (6.5), because here all matrices are known. For Newton iterations the full Jacobian has to be computed previously.

Nevertheless, in this chapter the linearized systems (6.5) and (6.8) are solved by the GMRES method proposed by Saad and Schultz (1986). In order to solve those non-symmetric linear systems, GMRES minimizes the norm of the euclidean residual vector $\|A\vec{x} - \vec{b}\|_2$ based on a Krylov subspace \mathcal{K}_m . In each iteration, the norm of the residual $r_m = \|\vec{r}(\vec{x}_m)\|_2$ with the current solution vector \vec{x}_m is calculated. The solution vector can be determined using a linear combination of basis vectors, where $\vec{x}_m = \vec{x}_0 + \mathcal{K}_m(A, r_0)$.

For constructing an orthonormal basis $\{\vec{v}_1, \dots, \vec{v}_m\}$ of a Krylov subspace either an Arnoldi process (Arnoldi, 1951) or, as in our case, a Gram-Schmidt process (Leon et al., 2013) can be used. In most GMRES implementations the method is matrix-free, meaning only the action of a matrix on a vector is evaluated instead of storing the complete matrix. This is a significant advantage for the solution of (6.8).

If $A \in \mathbb{R}^{n \times n}$ then the exact solution is reached in $m = n$ GMRES iterations, because the residual based on subspaces cannot increase. In each iteration a basis vector $\vec{v} \in \mathbb{R}^n$ has to be stored. Therefore, for large equation systems it is inefficient to save all Krylov search directions until convergence is reached. As a result, the restarted GMRES(m) algorithm was developed. Here, GMRES is restarted with the last search direction of the iterative process. The history of search directions is lost which leads to free memory but as a drawback, GMRES loses its guaranteed convergence properties. Later in this chapter, the influence of the restart parameter m onto the convergence of our solver will also be evaluated.

For the Picard iteration method, GMRES is applied to the linearized system (6.5). The main advantage of such methods is that only the action of the operator matrix A on a vector \vec{v} is required. In contrast, the Newton method is used in connection with a GMRES as a Newton-GMRES method, see Kelley (2003). Here only the action of

the Jacobian $J(\vec{x}^n)$ on a vector \vec{v} has to be calculated. Therefore, the finite difference quotient in one GMRES iteration reads

$$J(\vec{x}^n)\vec{v} = \frac{\vec{r}(\vec{x}^n + \epsilon\vec{v}) - \vec{r}(\vec{x}^n)}{\epsilon} \quad (6.9)$$

with ϵ denoting the differencing increment. With this approach, the Jacobian has not to be stored and can be evaluated matrix-free.

6.2.3 Choice of preconditioning technique

In almost all applications for saddle-point problems GMRES is used with a sufficient preconditioner in order to increase its convergence speed. The overall aim of a preconditioner is to transform the linear system to another linear system with better properties for solution algorithms like GMRES. Thus, by multiplying the complete equation system including RHS with a preconditioning matrix P , the overall system will have a smaller spectral condition number. Therefore, GMRES will converge in a significantly smaller number of iterations than without preconditioning. For more information about preconditioning of saddle point problems in general, see Benzi et al. (2005).

In this work the left-preconditioned approach by multiplying the matrix P^{-1} from the left with the equation system is being used. The linearized equation system within GMRES will be simplified by $A = A(x^n)$ and $F = F(u^n)$ in this subsection to increase readability:

$$P^{-1}A\vec{x} = P^{-1}\vec{b}. \quad (6.10)$$

The main difference in comparison with the right-preconditioned approach is, that the termination residuum is the one of the preconditioned system $\|P^{-1}\vec{r}(\vec{x})\|_2 = \|P^{-1}(A\vec{x} - \vec{b})\|_2$.

Note that, depending on which method is used to treat the nonlinearity, the GMRES algorithm slightly differs. Thus, the requirements on good preconditioning techniques are different. For the Picard linearization the preconditioning matrix should be a good approximation to the inverse of the linearized saddle point matrix out of (6.5), whereas for Newtons method, the inverse of the Jacobian has to be approximated, see (6.8). However, the approximation of the inverse of the saddle point matrix is usually also a good approximation for the Jacobian. Therefore, there will be no distinction between preconditioners depending on which technique is used for linearization.

In the following, the preconditioners evaluated in this work will be introduced. They will be compared in numbers of iterations first. Later on, the best preconditioners are evaluated on their computational performance and parallel efficiency. This will be part of the subsequent sections.

6.2.3.1 Schur methods

For deriving the Schur-type methods, a simple LU factorization of the saddle point system (6.5) is done:

$$\underbrace{\begin{pmatrix} I & 0 \\ BF^{-1} & I \end{pmatrix}}_L \begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \vec{u} \\ p \end{pmatrix} = \underbrace{\begin{pmatrix} I & 0 \\ BF^{-1} & I \end{pmatrix}}_L \begin{pmatrix} \vec{f} \\ g \end{pmatrix}, \quad (6.11)$$

which leads to

$$\underbrace{\begin{pmatrix} F & B^T \\ 0 & -(BF^{-1}B^T) \end{pmatrix}}_U \begin{pmatrix} \vec{u} \\ p \end{pmatrix} = \begin{pmatrix} \vec{f} \\ g - BF^{-1}\vec{f} \end{pmatrix}. \quad (6.12)$$

Here, $S = (BF^{-1}B^T)$ denotes the Schur complement. For solving, the system (6.14) is solved decoupled, meaning (6.12) is solved by two systems, which denote as

$$Sp = -g + BF^{-1}\vec{f} \quad (6.13a)$$

and

$$F\vec{x} = \vec{f} - B^T p, \quad (6.13b)$$

where S has to be inverted only in the pressure space, whereas F has to be solved in velocity space. Note that if $\vec{f} \neq 0$, another linear equation solve is required.

During this work, the Schur complement (6.14) is used as a preconditioner P and therefore multiplied with the system matrix A like it can be seen in (6.10). If matrix A is right preconditioned by U^{-1} , the eigenvalues of the matrix L are all identically 1 and the GMRES algorithm would need only two steps to converge. More details can be found in Elman et al. (2014). However, because F has to be still inverted, it is not feasible to use the exact Schur complement matrix especially for preconditioning. This leads to a matrix which approximates the convection-diffusion part F by M_F and the Schur complement $-(BF^{-1}B^T)$ by M_S :

$$\begin{pmatrix} F & B^T \\ 0 & -(BF^{-1}B^T) \end{pmatrix} \approx \begin{pmatrix} M_F & B^T \\ 0 & -M_S \end{pmatrix} = P. \quad (6.14)$$

Now, the crucial part is to choose a good approximation M_F and M_S , such that the computational effort is as small as possible but the convergence speed is as fast as possible.

SIMPLE method

According to Elman et al. (2014) the semi-implicit method for pressure linked equations (SIMPLE) method (Patankar and Spalding, 1972) is similar to the triangular block factorization of (6.12). In the SIMPLE method M_S is denoted as follows:

$$M_S = (B\hat{F}B^T) \quad (6.15)$$

In most common SIMPLE implementations, \hat{F} is chosen to be the diagonal of F and M_F^{-1} is a good approximation on F^{-1} , usually determined via multigrid cycles. For small calculations this can be done by a direct solver and was published for the underlying DG discretization in Klein et al. (2012). In this work, M_F is chosen to be $M_F = F$ for all Schur-type preconditioners, such that only the approximation of the Schur complement is evaluated.

Schur

Instead of using just the diagonal of F as an approximation for F in M_S , there are two other approaches in literature. The first one is the so called pressure convection-diffusion preconditioner published by Kay et al. (2002) and Silvester et al. (2001). For more details see those references.

In this work the least square commutator preconditioner developed by Elman et al. (2006) is used. Here, M_S is approximated as follows:

$$M_S = (BT^{-1}B^T)(BT^{-1}FT^{-1}B^T)^{-1}(BT^{-1}B^T). \quad (6.16)$$

T is the diagonal of the mass matrix, which in this solver is modified to be the identity. Thus, solving (6.13a) requires two Poisson-type solves and several matrix-vector products, which are negligible. The Poisson-type systems are solved by the direct solver MUMPS.

6.2.3.2 Schwarz methods

Schwarz domain decomposition methods are parallel, fast and robust algorithms for the computation of linear equations. Especially for the application in terms of preconditioning for Krylov-subspace methods, domain decomposition is widely used. In the following, the basic domain decomposition methods used in this work are introduced:

The basic idea is to split the total linear equation system into smaller parts and solve them with, for example, a direct solver. Afterwards, the solution of those smaller problems is used to correct the global solution and minimize the residual. Let R denote the restriction matrix, which, applied to a vector, returns only the values in a particular domain. In contrast, R^T is called the interpolation matrix, which does the opposite and scales a smaller local matrix up to the global matrix with its specific entries and zero elsewhere.

Therefore, the best local correction of a linear equation system $A\vec{x}^n = \vec{b}$ is derived to be

$$\text{correction} = R^T(RAR^T)^{-1}R(\vec{b} - A\vec{x}^n) = A^{-1}(\vec{b} - A\vec{x}^n). \quad (6.17)$$

Here, RAR^T denotes the subblock of A with a certain partitioning. For further information see, e.g. Smith (1997).

For the small block problems all direct and iterative solvers can be applied. In this work the direct solver MUMPS (Amestoy et al., 2000) is used for the block solve, if not stated otherwise.

Block-Jacobi

First, a classical Block-Jacobi method exists. This has the advantage, that it works perfect parallel. For preconditioning A is subdivided into subproblems, which will be solved without coupling between the blocks for preconditioning, see:

$$P_{AS}^{-1} = \begin{pmatrix} A_1^{-1} & 0 & 0 \\ 0 & A_2^{-1} & 0 \\ 0 & 0 & A_3^{-1} \end{pmatrix} = R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2 + R_3^T A_3^{-1} R_3, \quad (6.18)$$

whereas R_i denotes the restriction or interpolation matrix for the i -th block. A schematic representation of such a Block-Jacobi matrix can be found in Figure 6.4.

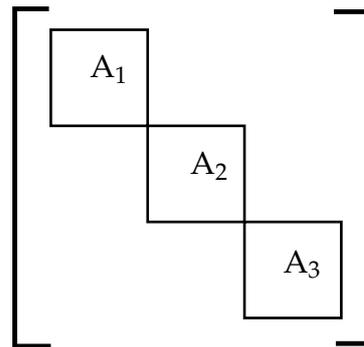


Figure 6.4: Schematic representation of matrix A consisting out of three different submatrices.

Overlapping domain decomposition

The Block-Jacobi is easy to parallelize, but because the blocks are solved separately without coupling, the approximation to A^{-1} is not accurate enough for preconditioning. Thus, there are methods which use a so called overlapping domain decomposition to couple different blocks to each other. The approximations to the inverse of the operator matrix is significantly better. However, communication between the blocks is necessary which makes the efficient implementation in parallel more challenging. Here, the blocks are mainly determined by their geometric relation in the computational grid instead of how the DoF are stored in the vectors, as it is the case for most Block-Jacobi methods.

For an Additive-Schwarz method, as it has been used in this work, the blocks are divided into overlapping regions containing roughly the same number of DoF. Therefore, the preconditioner is given by

$$P_{AS}^{-1} = \sum_{i=1}^p R_i^T A_i^{-1} R_i, \quad (6.19)$$

where R_i simply returns the coefficients for the i -th overlap region. In the calculations of this work, an overlap width of 1 cell is chosen based on the significantly better approximation of A^{-1} . Note that the more overlap the better the approximation on A^{-1} , but the worse is the parallelization due to communication. A schematic representation can be found in Figure 6.5, where the overlap region can be seen.

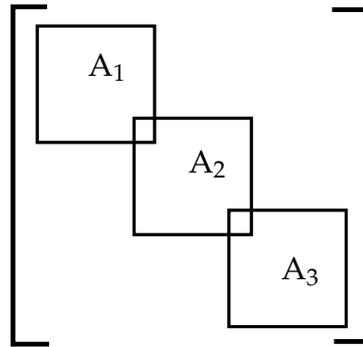


Figure 6.5: Schematic representation of matrix A consisting out of three different submatrices with overlap.

For an explanation of the multiplicative Schwarz method for overlapping domains the work of Smith (1997) is recommended.

Multigrid blocks

In case of multigrid blocks for preconditioning, the inverse of P will be approximated by calculating coarse grid solutions to A using a similar strategy as for classical multigrid. Therefore, the inverse of the preconditioning matrix can be written as follows:

$$P_{AS}^{-1} = \mathcal{I}_H \sum_{i=1}^n A_{C_i}^{-1} \mathcal{I}_h. \quad (6.20)$$

Here, \mathcal{I}_H is the restriction operator from the coarsest to the finest grid and \mathcal{I}_h the prolongation operator. In the final multigrid level, each cell is solved as a block with overlap like it has been demonstrated in (6.19). Therefore, n denotes the number of cells in the coarse grid. The cell local inverse of A^{-1} is determined at the final multigrid level and then prolonged up to the finest grid to yield a global approximation.

According to Smith (1997) the multilevel Schwarz method has very good convergence properties mostly independent of the problem size and number of processors. Note that for solving the subproblems of the coarsest grid an arbitrary solver can be used.

In large calculations it is rewarding to use an approximative iterative method for subproblems.

Coarse-grid solution

Additionally, a complete coupled solve for the coarsest multigrid level is applied to accelerate convergence for all Schwarz preconditioner methods. This leads to a significant improvement for the accuracy of P^{-1} . As it can be seen in (6.21), the coarse solution can be built by inverting the complete coarse operator matrix using the direct solver MUMPS in all calculations.

$$P_{MG}^{-1} = \mathcal{I}_H A_{\text{coarse}}^{-1} \mathcal{I}_h. \quad (6.21)$$

Afterwards, the total approximation consist of a coupled coarse part P_{MG} and a decomposed part P_{AS} , which can be seen in (6.22).

$$P^{-1} = P_{MG}^{-1} + P_{AS}^{-1}. \quad (6.22)$$

6.3 Precondition benchmarks

This section aims to compare the number of iterations GMRES needs by applying different preconditioning techniques to reach a linear residual of $1E-5$ in the last nonlinear iteration for a stationary problem. The comparison is similar to the one made by Elman et al. (2014). The non-restarted GMRES is linearized using a Picard or a Newton technique and then a GMRES or Newton-GMRES is applied to solve the linearized problem, respectively. Usually, for large calculations this is not feasible, because all search directions have to be saved which leads to a lack of memory on most systems. A more common approach is to restart the GMRES algorithm after saving m search directions with the current solution vector. However, for the restarted case GMRES loses its direct solver properties. The restarted GMRES is tested in the end of this section for the best preconditioner evaluated previously.

All preconditioners are used as it has been described Section 6.2. For the Schwarz domain decomposition preconditioners, fixed block sizes of 1000 (AS-1000), 5000 (AS-5000) and 10000 (AS-10000) DoF per block are chosen. This is because systems of up to 10000 DoF are considered to be easily solvable by a direct solver. The block partitioning is created by using the software METIS (Karypis and Kumar, 1998) and the overlap level is set to 1 for all calculations.

The total number of multigrid levels is set to 3 for all Schwarz calculations. Thus, all Schwarz calculations are improved by using a coarse solver on the coarsest multigrid level to couple all block information together, see (6.21). If a Schwarz preconditioner is used together with the multigrid blocks approach (AS-MG) the blocks are formed on the coarsest multigrid level. Here, the cells on the coarsest level are denoted as blocks. Ending up with a significant smaller block size and a larger number of blocks in total.

Here, (6.20) is solved until a multigrid depth of 2 and again with an overlap of 1 for the coarsest blocks.

In this section, first the results for boundary fitted testcases of a channel flow are presented. Afterwards, a particle will be immersed in the fluid flow to evaluate the influence of cut cells on the overall number of iterations. For both combinations, two and three dimensional cases are investigated. Although the channel flows are basically a linear testcase, it is a good indication how the convergence of the solvers change, if nonlinear effects like convection come into play.

Table 6.1: Computational settings for preconditioning tests with channel flow and immersed cylinder/sphere meshes.

Setting No.	Domain	Cells	DoF per cell	DoF total
1	$(-0.5,1.5) \times (-0.5,0.5)$	76×38	26	75 088
2	$(-0.5,1.5) \times (-0.5,0.5)$	128×64	26	212 992
3	$(-0.5,1.5) \times (-0.5,0.5) \times (-0.5,0.5)$	$18 \times 13 \times 13$	70	212 940

In Table 6.1 all settings are listed. For every calculation made in this section the polynomial degree $k = 3$ for velocity and $k = 2$ for pressure has been used. For all boundary fitted calculations a channel flow is considered. In the 2D case a parabolic velocity inlet with $\vec{u} = 1 - y^2$ at $x = -0.5$, pressure outlet with $p = 0$ at $x = 1.5$ and no-slip wall boundary conditions at the other boundaries are imposed. For 3D calculations all boundary conditions are imposed in the same spatial directions. Additionally, a periodic boundary condition is imposed in z -direction. As a result, this renders the flow to be between two infinite plates. Initially, the flow is at rest at $t = t^0$.

For the immersed boundary calculations a cylinder (2D) or sphere (3D) is centered at the origin of the domain. The radius is set to $r = 0.1$ and no-slip boundary conditions are applied at the surface. As a drawback, the blocking ratio between the immersed body and the channel walls is not optimal. However, the key results of this section are comparing various preconditioning techniques neglecting physical values like drag or lift coefficients. All calculations are evaluated using viscosities from $\mu_f = \{1, 0.1, 0.01\}$ resulting in increasing Re.

6.3.1 Boundary fitted

First, a simple two dimensional channel flow is evaluated. As already mentioned, different preconditioners are evaluated with different viscosities. The numbers in the following tables indicate the linear GMRES iterations needed to converge in the last nonlinear iteration. In addition, the number of nonlinear iterations in total is put into brackets. To increase readability, the best result is highlighted using bold numbers.

In Table 6.2 it can be seen, that for the small testcase the nonlinear iterations needed do not differ much between Picard and Newton linearization techniques. Only for the case of $\mu_f = 0.01$ a notable difference of one nonlinear iteration occurs. For both linearization techniques Additive-Schwarz preconditioners gain better results than the

Schur type preconditioners. Focusing on the Schur method, the fully resolved Schur complement is superior in comparison with the less qualitative approximation of the convective term in the SIMPLE approach. Also as expected, a growing block size leads to better results for the Additive-Schwarz type methods. Comparing the linearized systems, the Newton-GMRES needs less iterations than the GMRES combined with Picard linearization in almost all cases.

Table 6.2: Number of total GMRES iterations for setting 1 (2D, 75 088 DoF) with channel flow.

Picard						
μ_f	Schur	Simple	AS-1000	AS-5000	AS-10000	AS-MG
1	70 (2)	301 (2)	30 (2)	15 (2)	12 (2)	25 (2)
0.1	64 (2)	269 (2)	35 (2)	20 (2)	16 (2)	29 (2)
0.01	90 (3)	313 (3)	93 (3)	58 (3)	56 (3)	81 (3)
Newton						
μ_f	Schur	Simple	AS-1000	AS-5000	AS-10000	AS-MG
1	65 (2)	256 (2)	22 (2)	10 (2)	9 (2)	18 (2)
0.1	65 (2)	235 (2)	24 (2)	13 (2)	11 (2)	20 (2)
0.01	7 (3)	264 (2)	75 (2)	48 (2)	46 (2)	66 (2)

Increasing the amount of DoF significantly leads to some interesting effect, see Table 6.3. The amount of nonlinear iterations stays the same for both techniques as well as the tendency that Schwarz methods still work better than Schur-type methods. However, in contrast to the Schwarz-type methods, the Schur methods gain a lot of iterations comparing them with Table 6.2. Note that the number of cells per Additive-Schwarz block stays the same as in the previous case. However the problem is split into more blocks.

Table 6.3: Number of total GMRES iterations for setting 2 (2D, 212 992 DoF) with channel flow.

Picard						
μ_f	Schur	Simple	AS-1000	AS-5000	AS-10000	AS-MG
1	102 (2)	436 (2)	17 (2)	11 (2)	10 (2)	15 (2)
0.1	93 (2)	390 (2)	20 (2)	13 (2)	12 (2)	15 (2)
0.01	133 (3)	455 (3)	63 (3)	54 (3)	52 (3)	53 (3)
Newton						
μ_f	Schur	Simple	AS-1000	AS-5000	AS-10000	AS-MG
1	93 (2)	415 (2)	12 (2)	8 (2)	8 (2)	11 (2)
0.1	95 (2)	318 (2)	14 (2)	9 (2)	9 (2)	10 (2)
0.01	5 (3)	15 (3)	51 (2)	40 (2)	40 (2)	41 (2)

Extending the problem to three dimensions in setting 3 by keeping the total amount of DoF the same leads to less cells than in the previous case, but the DoF are stronger coupled. Interestingly, the Schur-type methods have a different behavior depending

on the linearization method and the viscosity. In almost all cases those methods work better for setting 3. In contrast, if the viscosity is set to $\mu_f = 0.1$, the Schur methods work worse for Picard and better for Newton linearization in three dimensions.

Table 6.4: Number of total GMRES iterations for setting 3 (3D, 212 640 DoF) with channel flow.

Picard						
μ_f	Schur	Simple	AS-1000	AS-5000	AS-10000	AS-MG
1	55 (2)	243 (2)	41 (2)	35 (2)	33 (2)	44 (2)
0.1	77 (3)	235 (3)	66 (3)	57 (3)	54 (3)	69 (3)
0.01	136 (3)	246 (3)	186 (3)	172 (3)	162 (3)	195 (3)
Newton						
μ_f	Schur	Simple	AS-1000	AS-5000	AS-10000	AS-MG
1	48 (2)	185 (2)	27 (2)	25 (2)	22 (2)	30 (2)
0.1	54 (2)	178 (2)	37 (2)	32 (2)	28 (2)	39 (2)
0.01	16 (3)	270 (2)	130 (2)	115 (2)	107 (2)	141 (2)

All in all, the boundary fitted tests render an Additive-Schwarz preconditioner to be the better alternative in all cases, although the Schur preconditioners work quite well for the three dimensional testcase with Newton linearization. Especially when it comes to parallelization, the domain decomposition approaches have an advantage which will be explored in Section 6.4.

6.3.2 Immersed boundary

For calculations with immersed particles the comparison of preconditioners will be restricted to Additive-Schwarz ones. This is based on the results obtained for the body fitted calculations in which the Schwarz preconditioners were completely superior.

Now, a cylinder for settings 1-2 and a sphere for setting 3 is immersed in the domain. This leads to more convective dominated flows, especially if the viscosity is decreased and thus the Re will be increased. From beginning to end of this section all settings have an increasing complexity for the solver. Furthermore, by the use of cut cells, the condition number of the operator matrices increases, although the agglomeration technique is applied for all simulations.

In Table 6.5 a small testcase with an immersed cylinder will be investigated. In comparison with the channel, the number of iterations for all nonlinear methods increase significantly with Re. This result is due to the fact that the convective terms in the NSE gain importance and so the nonlinear character of the overall system increases.

It is known that the Newton linearization leads to quadratic convergence whereas the Picard linearization only converges with first order. This can also be extracted from cases with $\mu_f = 0.01$. Here, Newton only needs 4 nonlinear iterations, whereas for Picard 8 iterations are needed to converge. As a result, Newtons method is even more superior in convection dominated flows. Again, increasing block size is beneficial for the number of linear iterations of the GMRES method.

Table 6.5: Number of total GMRES iterations for setting 1 (2D, 75 088 DoF) with immersed cylinder.

Picard				
μ_f	AS-1000	AS-5000	AS-10000	AS-MG
1	52 (3)	24 (3)	22 (3)	44 (3)
0.1	58 (4)	30 (4)	28 (4)	49 (4)
0.01	152 (8)	85 (8)	86 (8)	131 (8)
Newton				
μ_f	AS-1000	AS-5000	AS-10000	AS-MG
1	33 (2)	16 (2)	15 (2)	27 (2)
0.1	22 (3)	12 (3)	11 (3)	19 (3)
0.01	78 (4)	47 (4)	49 (4)	69 (4)

Table 6.6: Number of total GMRES iterations for setting 2 (2D, 212 992 DoF) with immersed cylinder.

Picard				
μ_f	AS-1000	AS-5000	AS-10000	AS-MG
1	27 (3)	23 (3)	20 (3)	31 (3)
0.1	32 (4)	29 (4)	26 (4)	34 (4)
0.01	92 (7)	86 (7)	81 (7)	102 (7)
Newton				
μ_f	AS-1000	AS-5000	AS-10000	AS-MG
1	18 (2)	14 (2)	13 (2)	19 (2)
0.1	13 (3)	11 (3)	10 (3)	14 (3)
0.01	49 (4)	47 (4)	46 (4)	51 (4)

After increasing the total number of cells in each direction, almost the same results can be gained, see Table 6.6. In contrast to Table 6.5 it is notable, that for both nonlinear solvers less linear iterations are needed. Surprisingly, also the number of nonlinear iterations for $\mu_f = 0.01$ using Picard decreases from 8 to 7. The reason for this is probably the better resolution of the problem in general.

In Table 6.7 the problem is extended to a three dimensional sphere immersed in a channel, which renders to be the most complex problem considered in this section. The comparison with the body fitted case leads to the same tendency as in two dimensions: An increasing amount of nonlinear iterations is needed due to the convection domination and the Additive-Schwarz preconditioner with the largest block size leads to the best results. Note that only 6 nonlinear iterations are needed for a Picard linearized system in three dimensions. It is to conclude, that the Additive-Schwarz preconditioner with large blocks is the best method of all methods considered.

Next, the solution strategy for setting 3 with an immersed sphere will be extended to a more practical one. As it was already described before, it is not economical in terms of memory to store all solution vectors of every GMRES iteration. Therefore, GMRES is

Table 6.7: Number of total GMRES iterations for setting 3 (3D, 212 940 DoF) with immersed sphere.

Picard				
μ_f	AS-1000	AS-5000	AS-10000	AS-MG
1	73 (3)	68 (3)	63 (3)	78 (3)
0.1	90 (4)	85 (4)	79 (4)	95 (4)
0.01	235 (6)	216 (6)	212 (6)	245 (6)
Newton				
μ_f	AS-1000	AS-5000	AS-10000	AS-MG
1	45 (2)	42 (2)	39 (2)	50 (2)
0.1	36 (3)	33 (3)	32 (3)	41 (3)
0.01	99 (4)	90 (4)	84 (4)	113 (4)

restarted after a maximum of m Krylov-dimensions. The parameter m is a trade-off between memory and fast convergence. The Krylov-dimension is commonly chosen to be between $m = 10$ and $m = 50$.

Table 6.8: Number of total GMRES(m) iterations in the last nonlinear iteration for setting 3 (IB, 3D, 212 940 DoF) with immersed sphere.

Picard				
μ_f	$m=10$	$m=20$	$m=30$	$m=50$
1	74 (3)	68 (3)	66 (3)	65 (3)
0.1	109 (4)	93 (4)	88 (4)	85 (4)
0.01	485 (6)	372 (6)	328 (6)	293 (6)
Newton				
μ_f	$m=10$	$m=20$	$m=30$	$m=50$
1	n.c.	41 (2)	41 (2)	39 (2)
0.1	40 (3)	35 (3)	33 (3)	32 (3)
0.01	141 (4)	114 (4)	106 (4)	95 (4)

In Table 6.8 all calculations of setting 3 are made with the most promising preconditioner AS-10000. Further, Picard and Newton linearization techniques are chosen for different viscosities again. The Krylov-dimension in this study is varied within a range of $m = \{10, 20, 30, 50\}$. If the results are compared to the fourth column of Table 6.7, it can be seen that first of all there is no change in number of nonlinear iterations for different m . Nevertheless, the number of linear iterations is always higher than in the non-restarted case, which is the expected behavior. It is remarkable that for Newton and $\mu_f = 1$ the restarted case with $m = 10$ did not converge at all.

By increasing m , convergence can be reached faster for Picard and Newton iterations. If all calculations with $m = 50$ are compared with the non-restarted ones of Table 6.7, it becomes clear that for Newton linearized systems very good results can already be obtained by choosing $m = 50$. This is also true for the Picard system with viscosities

$\mu_f = 1$ and $\mu_f = 0.1$. However, for $\mu_f = 0.01$ it becomes clear that Newton-GMRES is absolutely superior, especially for convective dominated cases.

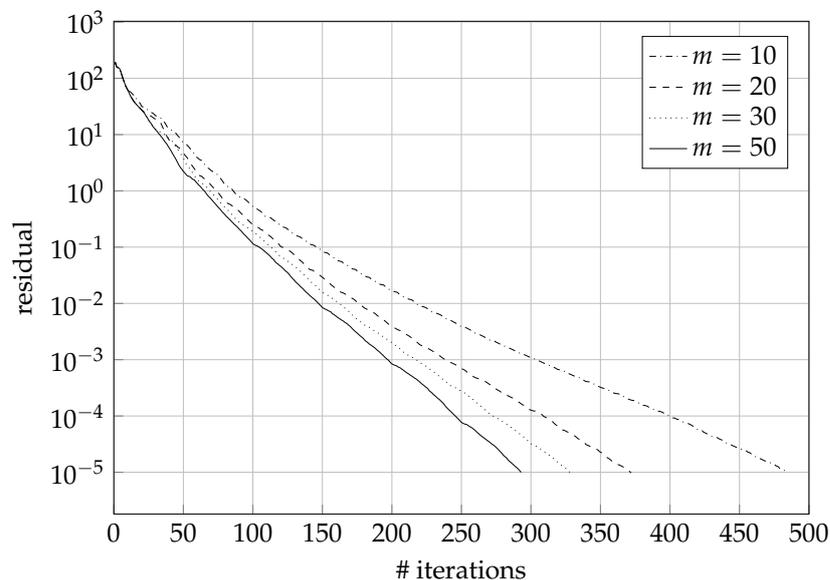


Figure 6.6: Picard using GMRES convergence with different m for $\mu_f = 0.01$ for setting 3 (3D, 212 940 DoF) with immersed sphere.

To further clarify the difference between Picard and Newton, the residuals of all GMRES iterations in the last nonlinear iterations are plotted in Figure 6.6 and Figure 6.7. In both figures it can be proven, that by increasing m the solver convergences faster (steeper slope). The starting residual of both last nonlinear iterations is surprisingly different. Whereas the preconditioned Newton-GMRES residual is already below 10^{-1} , the GMRES residual in the Picard case starts at above 10^2 and is therefore a reason for slow convergence. In contrast to the Picard linearization, a staircase pattern can be easily identified in the Newton case. This staircase pattern is typical for restarted GMRES convergence because after restart the slope is smaller.

Nonetheless, the gradient of the falling residual is also steeper in the Newton case. In Figure 6.6 a decrease of 10^{-1} needs around 50 linear GMRES iterations. A Newton linearized system can be decrease of the same amount by GMRES using less than 40 Newton-GMRES iterations.

All in all, Newton-GMRES with Additive-Schwarz preconditioning and a sufficient large block size points out to be the best iterative solver of all tested ones. In this context it has to be mentioned again, that only number of iterations were compared in this study. It is not possible to draw conclusions in terms of computational time from the tests above. A focus on computational time will be laid in the next section of this work. In addition, also only stationary problems have been investigated in this study.

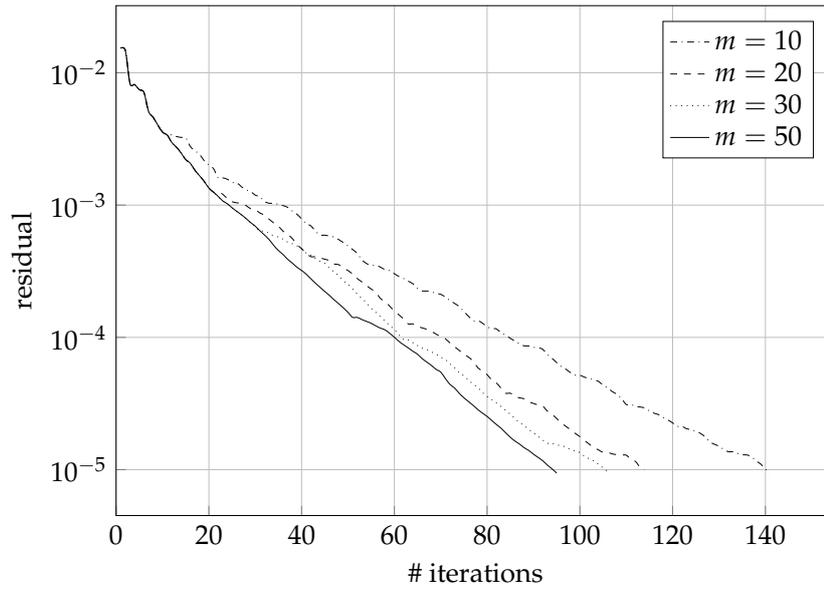


Figure 6.7: Newton-GMRES convergence with different m for $\mu_f = 0.01$ for setting 3 (3D, 212 940 DoF) with immersed sphere.

6.4 Performance analysis

Until now, only various equation solvers have been compared in terms of numbers of iterations. In this section, the performance of the method proposed in Section 4.2 will be analyzed. For this, a workflow tailored to the *BoSSS*-code will be presented. Then, a performance measurement will be carried out for both, single core and parallel cases, leading to a snapshot of the performance abilities at the date of this thesis. In the end, the most urgent hot spots will be identified.

Note that the main outcome of this work is the proposed workflow and the tuning using this workflow in Subsection 6.4.3. After those performance optimizations, the current bottlenecks are further analyzed in Subsection 6.4.4 and can therefore be tackled in the future.

6.4.1 Proposed workflow

The basic idea containing measurement, analysis and optimization is the same for all terms of performance, see Osterhage (2016). However, subsequently the explicit workflow tailored to *BoSSS* will be presented. In Figure 6.8, a graphical summary of the proposed optimization cycle can be seen.

In the following all steps will be explained in more detail. This contains also comments including practical experiences:

1. Take a measurement of the application using code instrumentation of the .NET-framework and get trace files serialized with JavaScript Object Notation (JSON) to be platform independent. This is particularly useful because performance

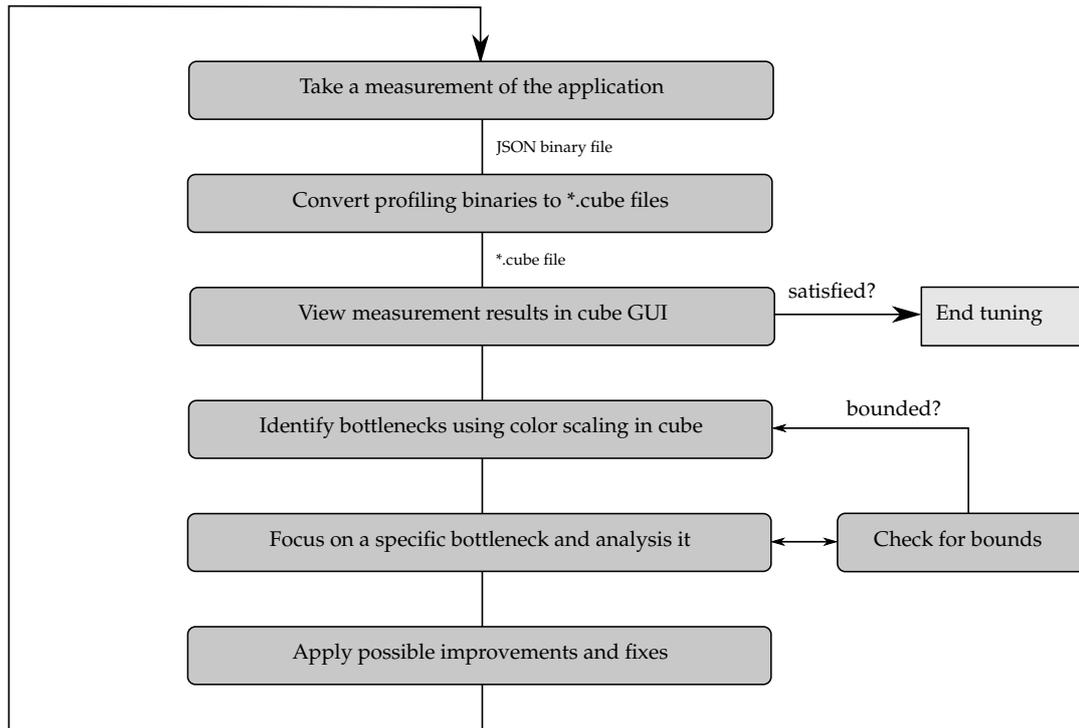


Figure 6.8: Proposed performance tuning cycle for usage in Subsection 6.4.3.

measurements can be performed on systems with Windows and Unix operating systems.

2. Convert profiling binaries to *.cube files using the *Cube* writer library (Saviankou et al., 2018). *Cube* is a performance report explorer tool developed at Jülich research center. For the conversion from *BoSSS* to *Cube*, a converter program was implemented using the writer library mentioned. This converter also supports profiling files of every processor in parallel runs and agglomerates them to a single *.cube file. For conversion, the metrics 'number of calls' and 'time' are currently supported.
3. View measurement results of the application by using *Cube* graphical user interface (GUI) (Saviankou and Cube developer community, 2018a,b). In *Cube*, the view of a call tree is present for every function. For parallel measurements, time and calls are listed for every processors. If satisfied with the performance of the code, end the tuning cycle.
4. Identify bottlenecks using *Cube* GUI. *Cube* uses a color scale to distinguish between hot spots and 'good' code. With this feature, also load balancing issues for parallel runs can be detected.
5. Focus on a specific bottleneck. Usually, it will be started with the most urgent hot spot. Nevertheless, it is a good idea to think about bounds in terms of performance, for example, by applying the roofline model (Ofenbeck et al., 2014) or time complexity analysis (Sipser, 2006). Maybe the most urgent hot spot

cannot be fixed easily because it is bounded by some other issues. If no bounds are known, check for the actual limiting factor.

6. Apply possible improvements and fixes to the code, recompile it and go back to step 1. Note that it is best practice to only apply one particular fix in order to evaluate the computational performance improvement.

All in all, this workflow for call count and time is simply applicable using the toolchain provided. However, if other metrics have to be considered, the *cube* converter tool has to be modified.

Further, it has been tested that also the performance modeling tool *ExtraP* (Calotoiu et al., 2013) can be used with the *.cube files created. *ExtraP* constructs mathematical models for the scaling behavior based on multiple measurements which are carried out for a given program. This is particularly useful, if it is compared with theoretical time complexity studies for a specific branch of the code. With *ExtraP* especially bottlenecks which probably occur with an increasing amount of processors will be detected in advantage.

6.4.2 Single core performance

All single-core calculations in this section have been performed using one core of the Intel Xeon on the local shared memory computing cluster of the chair of fluid dynamics at TU Darmstadt. The cluster has the following specifications:

- CPU: 4 Intel Xeon Processor E5-4627 v2 (8 cores in total)
- CPU-frequency: 3.30 GHz
- Cache size: 16 MB
- Shared Memory: 512 GB

Basically, the local computing cluster is one SMP node with 32 cores and access to a large amount of shared memory. Note that, because of other users at the chair of fluid dynamics, the cluster could not be used exclusively. However, for the results obtained quantitative statements can still be made.

Precondition test

At first, the precondition benchmarks of Section 6.3 have to be compared in terms of time. As already mentioned the comparison of time until a specific result is reached can deliver different results and conclusions than focusing only on the number of iterations. Therefore, the choice of the best preconditioner according to Section 6.3 has to be confirmed.

In Table 6.9 the column for calculations of $\mu = 0.01$ in Table 6.4 is evaluated in time using the presented workflow for both, Picard and Newton linearization techniques. Remember the configuration of setting 3 from Section 6.3 without an immersed particle.

For Picard as well as for Newton it becomes clear, that using an Additive-Schwarz domain decomposition technique is superior in comparison with Schur-type methods. It is also clear, that the system is solved faster with Newton because of less nonlinear iterations needed. However, an interesting outcome is that Schur preconditioning works faster for Picard linear systems. Surprisingly, the smallest block size of AS-1000 yields the fastest result although it needs more linear iterations than larger block sizes. This probably results out of the fact, that the block solver is faster in solving ten systems with 1000 DoF than in one system with 10000 DoF. Of course, this only shows the scaling of the direct solver MUMPS and might not be true for other block solvers.

Table 6.9: Time in 10^4 seconds for 3D Channel Flow with $\mu = 0.01$.

Linearization	Schur	Simple	AS-1000	AS-5000	AS-10000	AS-MG
Picard	6.36	11.0	2.87	3.65	4.87	3.48
Newton	7.14	9.38	1.32	1.62	2.11	1.62

Next, it will be checked if the tendency of the best preconditioner in view of solution time is also true for setting 3 with an immersed sphere. Like in Table 6.7, only Additive-Schwarz-type preconditioners are taken into account in Table 6.10.

Here, almost the same results have been reached for the test calculations. A system solve with Newton linearization is almost twice as fast as solving a system coming out of Picard linearization. Moreover, in contrast to the number of iterations the smallest block size with 1000 DoF is solved the fastest. Those results can be seen in Table 6.7.

Table 6.10: Time in 10^4 seconds for 3D Sphere Flow with $\mu = 0.01$.

Linearization	AS-1000	AS-5000	AS-10000	AS-MG
Picard	9.52	12.3	16.7	11.2
Newton	5.26	6.42	7.86	6.07

In the end of this part, it can be confirmed that Additive-Schwarz domain decomposition preconditioners are superior over Schur-type methods. Nevertheless, surprisingly the smallest block size solves the system the fastest. Thus, the convergence and the computing time of the Additive-Schwarz-type depends very sensitively on the size of the blocks.

Weak scaling

For checking the weak scaling behavior, a flow in a three dimensional driven cavity is used for benchmarking. The domain is $\Omega = (-0.5, 0.5) \times (-0.5, 0.5) \times (-0.5, 0.5)$ and to all walls a no-slip velocity boundary condition is applied. The upper wall moves with $\vec{u}_D = \{1, 0, 0\}$ and forces the flow to circulate inside the cavity. As initial condition velocity is set to $\vec{u}_0(x, y, z) = \{1, 0, 0\}$. The physical parameters of the fluid are chosen to be $\rho_f = 1$ and $\mu_f = 0.0025$ which renders down to $\text{Re}=400$. The problem is investigated on different Cartesian grids, leading to numbers of DoF from 4 000 to 500 000 using a polynomial degree of $k = 2$ for velocity and $k = 1$ for pressure.

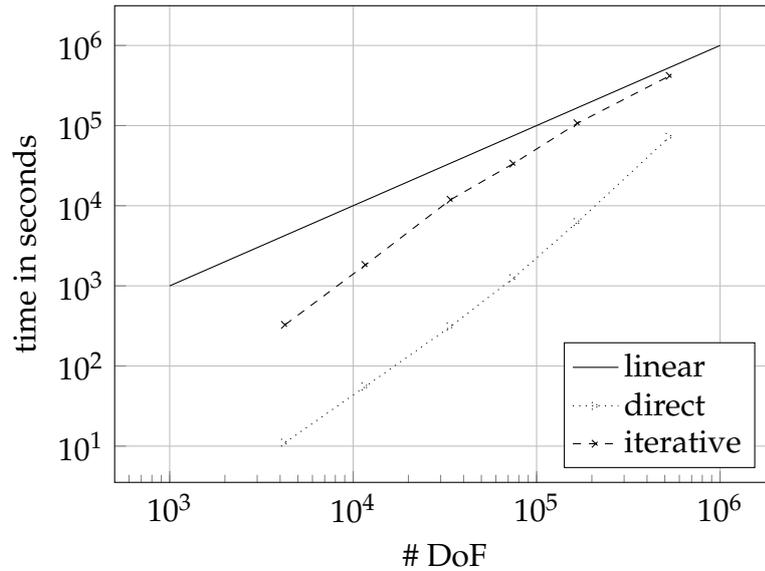


Figure 6.9: Increasing DoF for direct and iterative solver to evaluate weak scaling potential. Here the linear slope is an indication for perfect weak scaling. All slopes higher than linear do not fulfill weak scaling.

In Figure 6.9 two solution strategies for the driven cavity flow are compared. First, a Picard linearization technique is used with the direct solver MUMPS for the linearized system (direct). This is the fastest configuration for problems with a small number of DoF. Second, the Newton-GMRES approach is preconditioned with an iterative Additive-Schwarz preconditioner using a direct solver for each block and a coarse solve on the coarsest of three multigrid-levels (iterative). Of course, this means that the number of blocks and the size of the coarse system depends on the number of total DoF for each configuration.

Although the direct solver renders to be the best choice for all configurations, it can be seen from the slope approaching 1 Mio. DoF that there will be an intersection of both lines. This is because the direct solver does not scale linearly. The Newton-GMRES method reaches linear scaling for large amounts of DoF. Thus, it is more suitable for weak scaling than the direct solver.

6.4.3 Parallel performance tuning¹

The key goal of parallel analysis in this work is the measurement and tuning of parallel efficiency of the current solver. First, the focus lies on calculations with non-moving interfaces to track the worst scalability bugs for one timestep. All calculations in this section were made using MPI nodes of the Lichtenberg high performance cluster at TU Darmstadt. The SMP nodes used have the following configuration:

- CPU: 2 Intel Xeon Processor E5-2670 (8 cores)

¹All calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt.

- CPU-frequency: 2.60 GHz
- Cache size: 20 MB
- Shared Memory: 32 GB
- Node-Interconnect: Infiniband FDR-10

Note that for calculations up to 16 cores the node-interconnect for communication is not being used. However, in this section parallel efficiency means MPI parallelization instead of OpenMP.

In this section, two example issues in terms of parallel efficiency are presented. Further, they are focused using the tuning cycle in Subsection 6.4.1. In the end, the parallel efficiency of the current solver will be evaluated for non-moving configurations. This is sufficient, because by considering only stationary calculations with one timestep also the performance in moving-domain cases can be evaluated. A particulate flow simulation is multiple stationary simulations in a row, containing the creation of quadrature rules, evaluating quadrature, and solving the equation system. Note that because of the results of previous sections a Newton-GMRES algorithm is used with Additive-Schwarz preconditioning for all calculations in this section.

Block and coarse solve

In the beginning the focus is only laid on the equation solve. This usually is the largest part in the solver process. Especially, the Additive-Schwarz decomposition will be split into the solution of the coarse part which couples information over the whole domain together and the solution of the block part on each processors. Note that typically each block is assigned to one core for parallel scaling behavior.

If domain-decomposition is used, the solve on the coarsest multigrid level can only be performed on one core (neglecting OpenMP-parallelization). In contrast, the solution of blocks can scale almost perfect, except if overlap is present. Then, communication during the solving procedure has to be performed, but will be neglected in the following. In Figure 6.10, the idealized splitting of the solution process between sequential and parallel parts can be seen schematically.

Unlike the single-core performance it now matters which part of the calculation is only sequential and which part can be performed in parallel, see the law of Gustafson (1988) in (6.2). For the parallel case the law is denoted by (6.23).

$$S_{\text{Solver}} = \text{Coarse Solve} + p \cdot \text{Block Solve} \quad (6.23)$$

Here, the scaling of the solver is split into a constant sequential part and a scaling parallel part. It can easily be seen out of (6.2), that shifting the main computational part to best scaling parts of the code absolutely makes sense. By only shifting the computational load to the block solve and decreasing the time spent in the coarse solve, the parallel scaling of the overall method is raised from around 30% up to 70%.

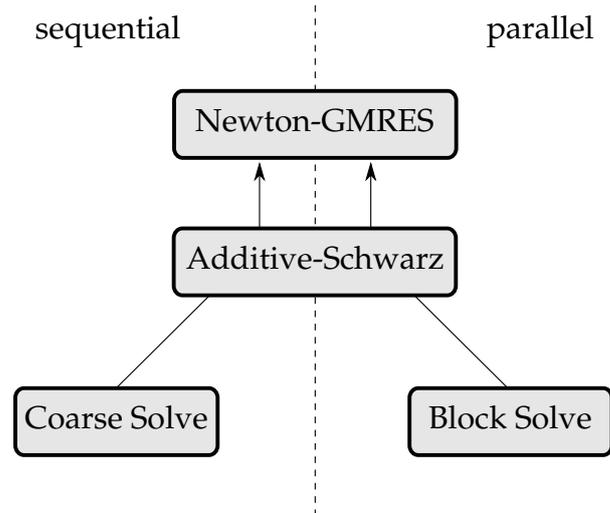


Figure 6.10: Sequential and parallel parts of equation solve process.

It is important to say, that shifts of the computational load to parallel functions can be a disadvantage for single-core performance. Remember the time spent in the overall solver might be constant for the single-core run, but if the system size of the coarse solve is decreased, the solver will also converge slower. This finally leads to worse results for the single core performance. Thus, optimization of parallel efficiency can be bad for single-core performance and vice versa. Therefore, it is important to firstly define goals of the current performance tuning process.

Load distribution

This part is about the importance of load distribution in order to obtain good parallel scaling. Especially for MPI-parallelization it is important to avoid idle times of processors and unnecessary communication. In general, it is not possible to find a 'perfect' load distribution for domain decomposition, which is best in terms of computational load and communication. Additionally, several parts of the code can have a different scaling behavior. Commonly, a trade-off of those two is required.

Here, the example of a three dimensional sphere flow from Section 6.3 is investigated for strong scaling behavior. However, setting 3 is modified to a new setting 4, which can be seen in Table 6.11. Setting 4 has a significant larger number of cells and is carried out with polynomial degree of $k = 1$ for velocity and $k = 0$ for pressure.

Table 6.11: Computational setting 4 extending setting 3 (3D, 212 640 DoF).

Setting No.	Domain	Cells	DoF per cell	DoF total
4	$(-0.5,1.5) \times (-0.5,0.5) \times (-0.5,0.5)$	$64 \times 16 \times 16$	13	212 992

The fluid viscosity is chosen to be $\mu_f = 0.002$ and the diameter of the immersed sphere to $d = 0.2$. This leads to $Re=100$. Boundary conditions are the same as for setting 3 in Section 6.2 and initial values for velocity and pressure are equal to zero in the complete computational domain.

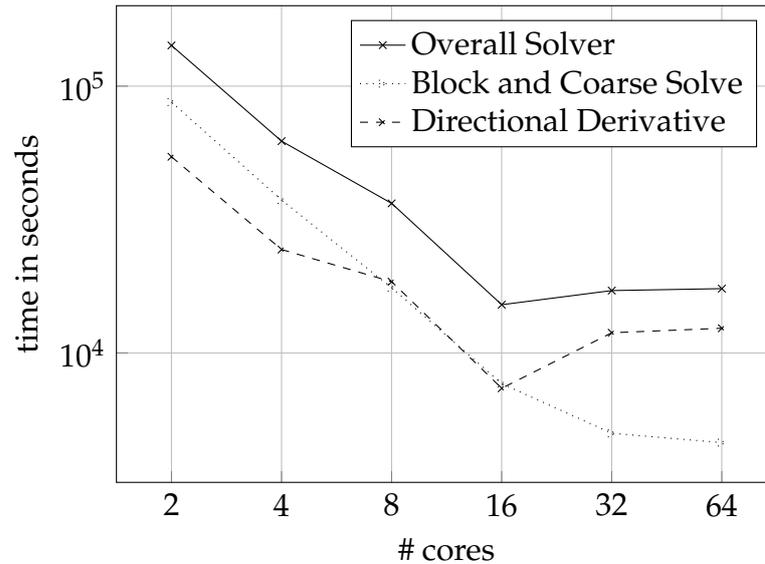


Figure 6.11: Strong scaling results for solver parts before tuning.

At first the performance of the overall solver was measured. In Figure 6.11, the time needed over a growing number of cores can be seen. The parallel scaling behavior up to a number of 16 cores is sufficient. After that, there is no more scaling present for the overall solver. This only occurs in settings with immersed boundaries, which resulted out of computations for single-phase flows.

For further investigation the Newton-GMRES equation solver is split up into the block and coarse solve part and the directional derivative. The block and coarse solve was already considered previously in this section, so its scaling is also sufficient, see Figure 6.11. The big lack of scaling seems to result from building the directional derivative in the Newton-GMRES algorithm. Remember that in (6.9), $\vec{r}(\vec{x}^n + h\vec{v})$ has to be evaluated in every GMRES iteration.

With usage of the proposed workflow, the bottleneck is analyzed and it becomes clear that the assembly and evaluation of the system matrix in the directional derivative does not scale properly due to a load imbalance on few cores. In detail, the quadrature on those cells cut by the interface of the sphere is only carried out by some of the cores if the number grows beyond 16. Using the domain decomposition strategy for the whole system finally creates some domains without and some including cut cells ('Standard'). Quadrature on cut cells is much more expensive, which leads to a significant load imbalance.

As the bottleneck is analyzed, different distribution strategies for the domain decomposition have been tested. First a distribution criterion is used, in which the cut cells are weighted by a factor of 50 in comparison with cells in the fluid phase ('1:50'). This still leads to some cores, containing only a few cut cells but more bulk cells and vice versa. The time difference in integration is therefore taken into account. Second, fluid and cut cells are equally distributed among all cores, leading to domains which are not connected, which is obviously bad for communication ('Equal').

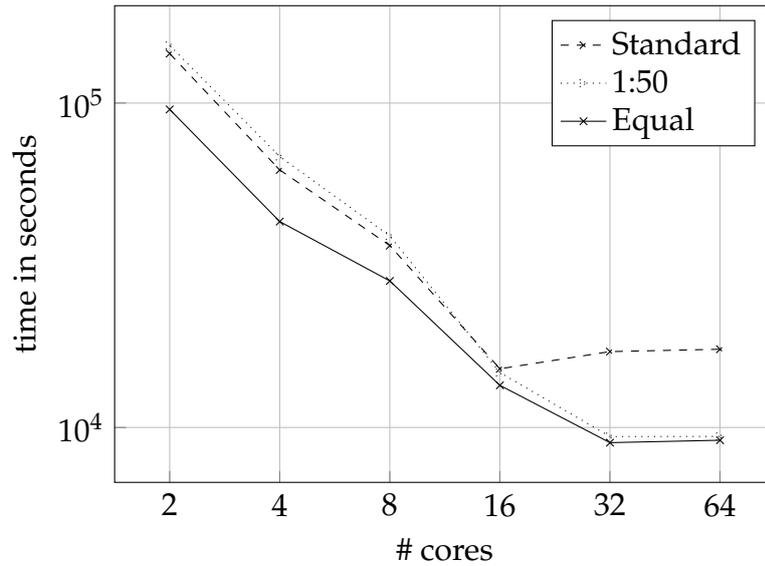


Figure 6.12: Strong scaling results for different cut cell distributions among cores.

In Figure 6.12 the results of the standard distribution and both other distributions for the overall solver can be seen. Clearly the standard distribution performs worst whereas both modified distributions deliver a better performance beyond 16 cores. For up to this number, the 1:50 distribution performs almost like the standard method. This is due to the contrary requirements of good parallel scaling for the solver and quadrature: The equation solve basically does not care about cut cells, only the number of total DoF on one core is crucial. In contrast, there exist a huge difference in the computational time needed for fluid in comparison with cut cells for quadrature.

The best load distribution is yield, if fluid and cut cells are equally distributed among the cores. This leads to good parallel scaling of the equation solve as well as for the quadrature in the directional derivative of Newtons method. Unfortunately, the communication expenses are the worst for this strategy. Nevertheless, it seems to be negligible for the small number of cores. A influence of communication can be observed for a growing number beyond 32.

In Figure 6.13 the parallel scaling of the current solver after tuning can be seen. Here, the best load distribution strategy of the previous investigating is being used together with the presented shift of computational effort to the best scaling functions. For computations only on one SMP-node, a good scaling value of 80% is reached. For a growing number of cores, the node-interconnect Infiniband FDR-10 network has to be used for communication, revealing the huge communication overhead which comes out of the equal distribution strategy.

Overall, the parallel efficiency of the solver is increased by a factor of more than two. However, for a growing number of cores, communication becomes more and more important. As a result alternatives to the load distribution strategies presented have to be investigated.

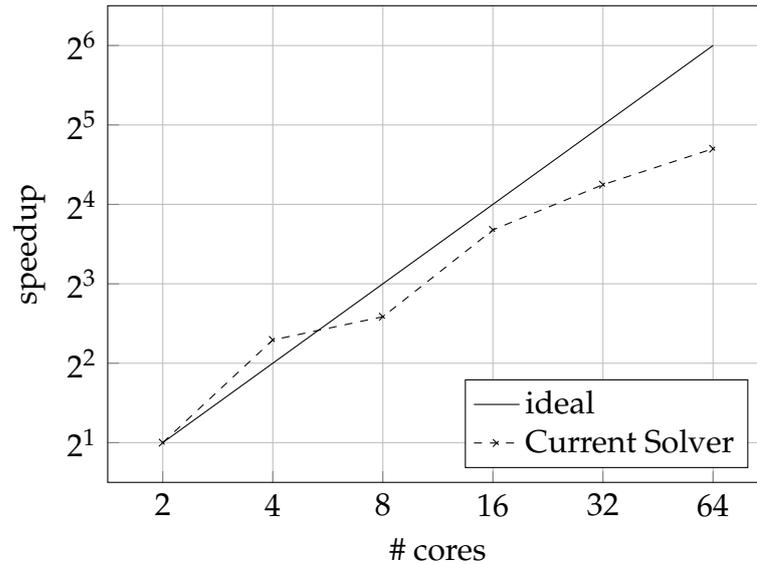


Figure 6.13: Overall parallel efficiency after tuning.

6.4.4 Current bottlenecks

In the previous parts of this section, single-core and parallel performance were evaluated. In addition some parallel scaling issues concerning solver configurations and load imbalances have been tackled. As the parallel scalability is satisfying, this part aims to point out the current bottlenecks of the code for boundary fitted, immersed boundary and particulate flow testcases.

The testcases are extracted from other parts of this thesis and were repeated for performance measurement. All three different cases can be extracted from Table 6.12. For both non-moving tests, 3D calculations exist, whereas for the particulate flow case only 2D results can be evaluated. Resulting in a larger amount of DoF for 3D, a Newton-GMRES iterative solver is used with an Additive-Schwarz domain decomposition for preconditioning. Note that only for the moving-boundary case a time dependent calculation is measured.

Table 6.12: Configurations for current bottleneck measurements.

Configuration	Testcase	Dimension	Solver	# Timesteps
Boundary fitted	Channel flow	3D	iterative	1
Immersed boundary	Sphere flow	3D	iterative	1
Particulate flow	Falling ellipse 45°	2D	direct	2000

Moreover it is obvious, that the time fractions in the following can differ depending on polynomial degree, Re, grid resolution, and equation system solver. However, which parts of the code tend to be possible bottlenecks and have to be optimized first stays the same.

Boundary fitted

First, the most simple numerical case of a three dimensional channel flow is measured. Here, no immersed boundary is present and therefore there exist no cut cells. As it can be seen in Figure 6.14, the main part of the computational expense is the solution of the Additive-Schwarz blocks.

Since the parallel scaling of the block solve was previously shown, this issue can be tackled by using multiple cores. As also already mentioned the coarse solve part is purely sequential, but is still needed. Evaluating quadrature is with 3.9% of almost no meaning for the overall performance. Creating quadrature rules in fluid cells is even less important.

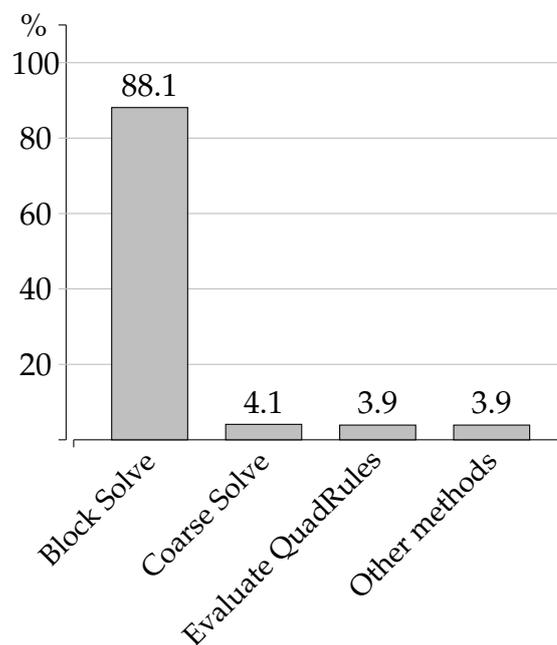


Figure 6.14: Percent of time spent in methods for 3D channel.

Thus, for boundary fitted calculations main bottlenecks can be tackled by running the system in parallel using domain decomposition. This procedure is able to speed up the solver significantly.

Immersed boundary

Now, a sphere is immersed into the channel leading to the appearance of cut cells in the computational domain. Additionally, the convective part of the NSE becomes of more importance.

Again, the solution of the blocks is the dominating part in the complete solution process. However, in contrast to the boundary fitted example, the time consumption of the quadrature grows. This is due to the fact, that creation and evaluation of quadrature rules in cut cells using HMF is a lot more expensive than those in pure fluid cells.

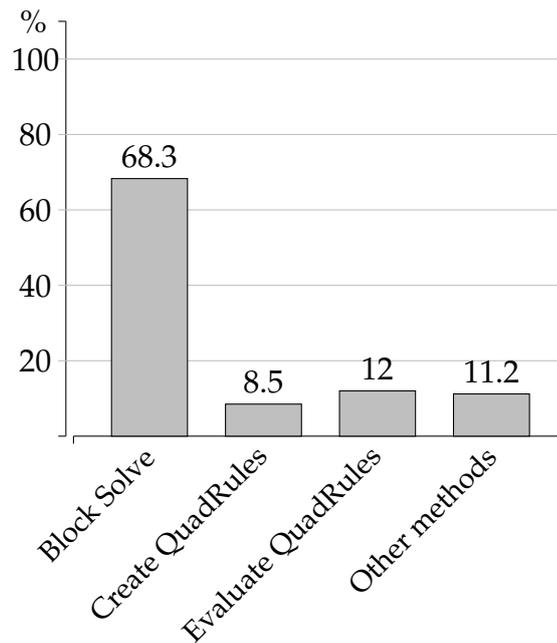


Figure 6.15: Percent of time spent in methods for 3D sphere.

If the flow around a sphere is computed for transient solutions with growing Re , the time spent in the creation of quadrature rules decreases. This is, because quadrature rules can be cached for further usage in the next time steps. Therefore, the solution of Additive-Schwarz blocks and the evaluation of quadrature will render to be the most consuming parts of the calculation for time dependent immerse boundary calculations.

Particulate flows

In the last example a fully coupled calculation of a falling ellipse is measured for bottleneck detection. In contrast to the first two examples, the time dependent calculation is carried out evaluating the flow and particle quantities at 2000 time steps. For the solution of the equation system a Picard linearization together with a direct solver is being used.

Figure 6.16 confirms the growing importance of quadrature for moving boundary calculations. Creation and evaluation of quadrature rules occupies almost half of the total computational time. This is due to the fact that the quadrature rules cannot be cached for proceeding time steps, they have to be recreated every step.

If the amount of particles in the fluid increases, the number of quadrature rules on cut cells does at the same time. In contrast, the total amount of DoF almost stays the same. Thus, the equation solve part stays constant and the creation and evaluation of quadrature part increases tremendously. This behavior cannot be tackled with parallelization techniques and distributions presented in this thesis, leading to the implementation of more sophisticated methods of distributing particles and computational domain over all cores available at the same time. Optimally, each core in a parallel computation is

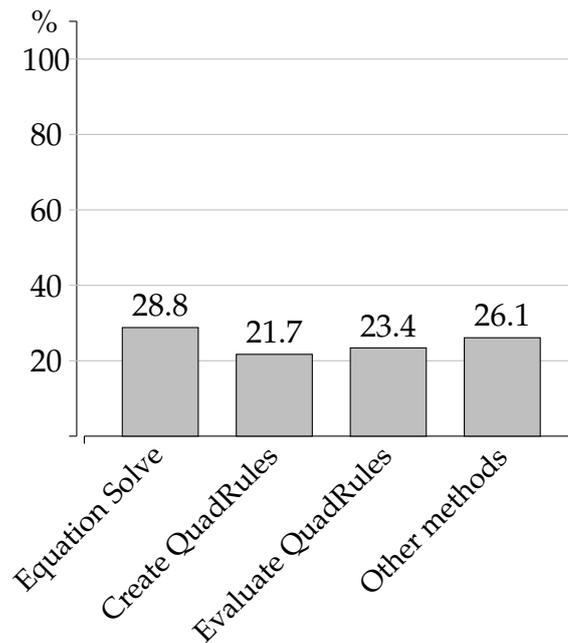


Figure 6.16: Percent of time spent in methods for falling ellipse 45° .

responsible for the creation and evaluation of quadrature rules for one particle in the computational domain. With this, parallel scaling can be achieved.

All in all, the solution of the equation system and the quadrature have to be addressed in future work to further increase the performance.

6.5 Conclusion

For the context of this chapter in the current work it first needs to be stated that iterative solvers only beat direct solvers for large 2D problems and 3D problems. Both were only touched partly for the validation tests in Chapter 5. Therefore the presented investigations are motivated by giving an outlook of extending the current method to larger problems in the future.

In the beginning of this chapter, state of the art approaches for the solution of the equation system are introduced as well as some basics about HPC and performance measurements. Afterwards, the process of solving the equations is analyzed in detail and possible preconditioners are tested for a GMRES method together with Picard and Newton linearization techniques. As a result the Newton-Krylov GMRES together with an Additive-Schwarz domain decomposition method performs best in terms of iterations. Here, the largest size of the domain blocks performs best.

Next, a performance analysis is carried out using a workflow proposed for the specific *BoSSS*-code. Afterwards, the Newton-Krylov solution technique for single-core calculations is tested by measuring time until convergence. In time perspective the smallest block size converges the fastest. However, in general the results of the previous section are confirmed.

In the following the weak scaling of the chosen iterative method is tested and is achieved for growing numbers of DoF, whereas the direct solver is superior for small problems. The parallelization of the whole algorithm is tuned by applying a shift of computational load to the best scaling functions and a 'smart' load distribution in terms of cut cells.

In the end current bottlenecks for boundary-fitted, immersed boundary, and particulate flow examples are presented. The key bottlenecks render to be the equation solve, creation and evaluation of quadrature rules. The solve part can easily be tackled by parallelization as shown before. For quadrature, especially for moving domains, some modifications have to be done in the future.

For further tuning it is possible to use a different quadrature method than the HMF, e.g. the one proposed by Saye (2016). Here, the rules for quadrature can be created much faster. However, the evaluation costs stay the same because almost the same number of nodes as in the HMF approach is used to obtain a similar accuracy. Another possible way is to tackle the bottleneck with further parallelization, where particles are distributed over processors. With this strategy, each core is responsible for the creation and evaluation of quadrature nodes of its own particle. In contrast to this, a so called Chimera-mesh can be used around the particles. Here, the creation of quadrature rules is only done once and the overlapping Chimera-mesh then is moved according to the particle motion. However, for solving NSE in the fluid domain, a projection between background and chimera mesh has to be performed every single particle movement.

All in all, a professional environment for the performance analysis of the *BoSSS*-code is proposed and both, single core and parallel efficiency, are analyzed pointing out current bottlenecks and possible solution strategies. Nevertheless, few performance analysis approaches are not considered in this work, e.g. boundedness is almost completely neglected. For this, a detailed theoretical time complexity analysis of the whole algorithm has to be performed to determine theoretical bounds. Typically, the roofline model is used (Ofenbeck et al., 2014) to detect hardware bounds in terms of memory- and/or CPU-performance.

To conclude the parallel efficiency of the code is sufficient. However, if compared to the methods of Wan and Turek (2007) and Uhlmann (2005) who investigate particulate flows up to 100,000 particles with low order discretizations, the proposed method renders to be computationally expensive. The tuning focus should be laid on single-core performance and implementation improvement in the future. Here an analysis of the theoretically possible peak performance will be necessary.

Please also note the possibility for developers to receive support from super-computing centers using their manpower and knowledge to tune code on a much higher-level. A possible workflow is proposed by Iwainsky et al. (2012). However, the cooperation with experts should be considered after further success in single-core performance tuning.

7 Conclusion and outlook

The overall goal of the proposed work was to develop a numerical method for particulate flow applications with various shapes with high-order accuracy and evaluate the computational efficiency afterwards. To fulfill this challenging task, first, a numerical method based on the cut cell DG approach was developed to account for high-order accuracy. For the representation of particle surfaces an immersed boundary method was used to avoid the cumbersome and time consuming re-meshing. Second, on top of that, a solver framework was introduced to analyze and solve resulting large systems of equations efficiently by using HPC clusters. The complete work was implemented into the open-source framework *BoSSS*, which is under active development at FDY at TU Darmstadt.

This thesis is split into three major parts, (i) the proposed numerical method using a cut cell DG approach is presented in detail, (ii) the method is validated considering various tests with increasing complexity, and (iii) the computational performance of the current solver is analyzed by proposing a general workflow for the *BoSSS*-framework and pointing out performance hot spots of the code.

Common methods assume particles to be mainly spherical, see Uhlmann (2005) and Wan and Turek (2006). They also preferably use low order schemes for discretization. However, the base of this work is the extended DG formulation proposed by Kummer (2016), which is a method of high accuracy. This solver was extended to simulate rigid particles of different shapes including a two-way coupling between fluid flow and particle motion using the common Lie-splitting approach for time discretization of moving domains.

For the mathematical model of fluid and particles, the incompressible NSE was used together with the NEE of rigid body motion. Further, the collision behavior was modeled by the conservation of momentum along the collision normals. All particle surfaces were assumed to be smooth. However, rotational momentum can be transferred due to eccentric collisions between two non-spherical particles and was taken into account.

The numerical discretization was based on a sharp interface DG approach, meaning the surface of the particles was described by using a characteristic function. For the convective term of the NSE, a local Lax-Friedrich flux and for the diffusion term, a standard SIP formulation was used. For the accurate integration on cut cells of arbitrary shape, the HMF method was used, see Müller et al. (2013) and Kummer (2016). In order to avoid bad condition numbers of the system, a so called cell agglomeration strategy was applied. Here, all fluid cells under a particular threshold value are agglomerated to their nearest neighbor before integration.

The coupling between fluid and particle worked by the integration of the Newtonian stress tensor over the particle surface and incorporating resulting forces into the NEE. Current translational and rotational velocities were treated for the fluid solver as boundary conditions at the particle surface. For the temporal discretization a BDF-2-scheme was used for the NSE and a Crank-Nicolson scheme of second order for the time resolution of the NEE.

The overall method was validated with various numerical experiments, from pure immersed boundary settings to fully coupled cases. For this, the results were compared to common methods for particulate flow simulations in literature. Moreover, strong correlations between the choice of the SIP penalty parameter and physical quantities like lift and drag forces were pointed out. The high-order convergence property of $\mathcal{O}(h^{k+1})$ for the spatial terms was shown therein, including the reduction of total DoF to reach the same accuracy as low order methods. However the Lie-splitting renders the method of first order in time only. Thus the overall method renders to be of $\mathcal{O}(\Delta t + h^{k+1})$. As a result the time step size has to be chosen small $\Delta t = h^{k+1}$ if polynomial degree increases in order to balance errors. Therefore to achieve a high-order method for moving domains a high-order time discretization alternative becomes necessary and should be subject of future research.

Collisions between particles and walls were modeled with a conservation of momentum or a repulsive force collision model. The collision detection was based on the knowledge which cut cells belong to which particle. By intersecting neighboring cells of those cut cells, the model was possibly triggered. The proposed detection algorithm was applied for arbitrary cut cells and is therefore perfectly suitable for the overall method.

Combined collision effects of two particles falling in incompressible fluid were evaluated with both collision models. Here, both led to the same results. However, because it is conservative by definition, the one based on momentum exchange is superior. Due to the lack of benchmark data in literature for flows with non-spherical particles, only qualitative comparisons were made. The behavior of particles with different shape was evaluated by testing the integration of hydrodynamical forces along a falling ellipse. Furthermore, the challenging test of five particles with different shapes falling in fluid was able to reproduce all expected physical phenomena like draft, kissing, tumbling, and fluid acceleration in small gaps due to wall collisions.

By applying the shape independent collision detection, also arbitrary geometries can be tracked for possible impacts. Through the use of a collision model based on conservation and the moving interface time discretization approach, the method is fully conservative. This further emphasizes the novelty of the presented work.

The proposed method was further extended to the three dimensional problem of a sphere flow due to numerous existing literature results for comparison. Good agreement with literature was obtained for the stationary ($Re=100$) sphere flow. However, the strong increase in DoF leads together with the more complicated quadrature on surfaces in three dimensions to a computationally challenging method especially in comparison with the work of Wan and Turek (2007), Glowinski et al. (2001) and

Uhlmann (2005). Therefore, a detailed performance analysis of the proposed method was needed.

Commonly, solving the nonlinear equation system is the most time consuming part of a CFD computation. Until now mostly a Picard linearization technique together with a direct solver has been used inside the *BoSSS* framework. For large equation systems the iterative solver GMRES with Schur-type and Additive-Schwarz preconditioners were tested. In addition to applying GMRES to the Picard system, a Newton-GMRES method was implemented. Several preconditioning combinations were tested in terms of number of iterations similar to Elman et al. (2014). Finally, the Newton-GMRES algorithm with Additive-Schwarz preconditioning and overlapping blocks leads to be absolutely superior in number of iterations and computational time.

In order to carry out reproducible and accurate performance measurements, a tuning workflow for the *BoSSS*-code was proposed. With this, performance measurements were visualized using the *Cube*-GUI (Saviankou and Cube developer community, 2018b). This workflow still delivers an important contribution to other users of the open source code *BoSSS*.

To increase parallel scalability, the main computational load of the solving procedure was shifted to the solution of the Additive-Schwarz blocks, which scale almost perfectly in parallel. Another main result was that the load distribution of cut cells is crucial for parallel efficiency. Therefore, in a proposed load balancing strategy, all fluid cells were distributed to all cores first. Afterwards, the same was done for all cut cells. This resulted in significant parallel efficiency of around 80% on one SMP-node. If the number of cores was further increased, the communication effects using the memory interconnect network dominated the scaling behavior. Nonetheless, the parallel efficiency was proven to be very promising even for a high number of cores.

Although the parallel efficiency is very promising, the overall computation times of low order methods of Wan and Turek (2007) and Uhlmann (2005) especially for a high number of particles are superior. This might be caused by the disadvantages of the DG method which leads to more DoF and larger stencils for system matrices.

At last the bottlenecks of the current solver were identified. For fully coupled calculations of a non-spherical particle the computational hot spots were denoted as follows: equation solve, creation of quadrature rules, and the evaluation of those rules. A smart load distribution strategy which distinguishes between solving the equation system and quadrature along particles can easily accelerate the current solver multiple times.

In conclusion, the entire work proposes a method for non-spherical particulate flow applications with high accuracy in space combined with Lie-splitting. In addition, the presented collision detection algorithm based on cut cells increases the versatility of the proposed method once again. As already stated, future research has to be focused on high-order time discretization which then renders the method to be of high-order for moving domains. For this, it should be focused on problems with few particles but high-order requirements to distinguish from common low order methods for particulate flows which perform well for large numbers of particles ($\geq 1,000$).

Bibliography

- Amdahl, G. M. (1967). Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, AFIPS '67 (Spring)*, pages 483–485, New York, NY, USA. ACM.
- Amestoy, P. R., Duff, I. S., and L'Excellent, J. Y. (2000). Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, 184(2):501–520.
- Antonietti, P. F., Sarti, M., Verani, M., and Zikatanov, L. T. (2017). A Uniform Additive Schwarz Preconditioner for High-Order Discontinuous Galerkin Approximations of Elliptic Problems. *Journal of Scientific Computing*, 70(2):608–630.
- Ardekani, A. M. and Rangel, R. H. (2008). Numerical investigation of particle–particle and particle–wall collisions in a viscous fluid. *Journal of Fluid Mechanics*, 596:437–466.
- Arnold, D. (1982). An Interior Penalty Finite Element Method with Discontinuous Elements. *SIAM Journal on Numerical Analysis*, 19(4):742–760.
- Arnoldi, W. E. (1951). The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9(1):17–29.
- Bassi, F., Crivellini, A., Di Pietro, D. A., and Rebay, S. (2007). An implicit high-order discontinuous Galerkin method for steady and unsteady incompressible flows. *Computers & Fluids*, 36(10):1529–1546.
- Bastian, P. and Engwer, C. (2009). An unfitted finite element method using discontinuous Galerkin. *International Journal for Numerical Methods in Engineering*, 79(12):1557–1576.
- Benzi, M., Golub, G. H., and Liesen, J. (2005). Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137.
- Bobrow, J. E. (1989). A Direct Minimization Approach for Obtaining the Distance between Convex Polyhedra. *The International Journal of Robotics Research*, 8(3):65–76.
- Bouma, W. and Vaněček, G. (1991). Collision Detection and Analysis in a Physically Based Simulation. In *Eurographics Workshop on Animation and Simulation*, pages 191–203, Vienna.
- Buyya, R., Cortes, T., and Jin, H. (2002). An Introduction to the InfiniBand Architecture. In *High Performance Mass Storage and Parallel I/O: Technologies and Applications*. IEEE.

- Calotoiu, A., Hoefler, T., Poke, M., and Wolf, F. (2013). Using automated performance modeling to find scalability bugs in complex codes. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12.
- Cameron, S. (1985). A study of the clash detection problem in robotics. In *1985 IEEE International Conference on Robotics and Automation Proceedings*, volume 2, pages 488–493.
- Cameron, S. and Culley, R. (1986). Determining the minimum translational distance between two convex polyhedra. In *1986 IEEE International Conference on Robotics and Automation Proceedings*, volume 3, pages 591–596.
- Campregher, R., Miltzer, J., Mansur, S. S., Neto, S., and Da, A. (2009). Computations of the flow past a still sphere at moderate reynolds numbers using an immersed boundary method. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 31(4):344–352.
- Cant, S. (2002). High-performance computing in computational fluid dynamics: Progress and challenges. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 360(1795):1211–1225.
- Chapelier, J. B., de la Llave Plata, M., Renac, F., and Lamballais, E. (2014). Evaluation of a high-order discontinuous Galerkin method for the DNS of turbulent flows. *Computers & Fluids*, 95(Supplement C):210–226.
- Chouippe, A. and Uhlmann, M. (2015). Forcing homogeneous turbulence in DNS of particulate flow with interface resolution and gravity. *Physics of Fluids*, 27(12):123301.
- Crivellini, A., D’Alessandro, V., and Bassi, F. (2013). High-order discontinuous Galerkin solutions of three-dimensional incompressible RANS equations. *Computers & Fluids*, 81(Supplement C):122–133.
- Culley, R. and Kempf, K. (1986). A collision detection algorithm based on velocity and distance bounds. In *1986 IEEE International Conference on Robotics and Automation Proceedings*, volume 3, pages 1064–1069.
- Dahlgren, F. and Torrellas, J. (1999). Cache-only memory architectures. *Computer*, 32(6):72–79.
- de Wiart, C. C., Hillewaert, K., Bricteux, L., and Winckelmans, G. (2015). Implicit LES of free and wall-bounded turbulent flows based on the discontinuous Galerkin/symmetric interior penalty method. *International Journal for Numerical Methods in Fluids*, 78(6):335–354.
- Dobkin, D. P. and Kirkpatrick, D. G. (1990). Determining the separation of preprocessed polyhedra — A unified approach. In Paterson, M. S., editor, *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 400–413. Springer Berlin Heidelberg.

- Duchanoy, C. and Jongen, T. R. G. (2003). Efficient simulation of liquid–solid flows with high solids fraction in complex geometries. *Computers & Fluids*, 32(10):1453–1471.
- Elman, H., Howle, V. E., Shadid, J., Shuttleworth, R., and Tuminaro, R. (2006). Block Preconditioners Based on Approximate Commutators. *SIAM Journal on Scientific Computing*, 27(5):1651–1668.
- Elman, H., Silvester, D., Wathen, A., and Wathen, A. (2014). *Finite Elements and Fast Iterative Solvers : With Applications in Incompressible Fluid Dynamics*. Oxford University Press.
- Elman, H. C., Silvester, D. J., and Wathen, A. J. (2002). Performance and analysis of saddle point preconditioners for the discrete steady-state Navier-Stokes equations. *Numerische Mathematik*, 90(4):665–688.
- Fadlun, E., Verzicco, R., Orlandi, P., and Mohd-Yusof, J. (2000). Combined Immersed-Boundary Finite-Difference Methods for Three-Dimensional Complex Flow Simulations. *Journal of Computational Physics*, 161(1):35–60.
- Fechter, S. and Munz, C.-D. (2015). A discontinuous Galerkin-based sharp-interface method to simulate three-dimensional compressible two-phase flow. *International Journal for Numerical Methods in Fluids*, 78(7):413–435.
- Feng, Z.-G. and Michaelides, E. E. (2004). The immersed boundary-lattice Boltzmann method for solving fluid–particles interaction problems. *Journal of Computational Physics*, 195(2):602–628.
- Ferrari, A., Munz, C.-D., and Weigand, B. (2010). A High Order Sharp-Interface Method with Local Time Stepping for Compressible Multiphase Flows. *Communications in Computational Physics*, 9(1):205–230.
- Fogelson, A. L. and Peskin, C. S. (1988). A fast numerical method for solving the three-dimensional stokes’ equations in the presence of suspended particles. *Journal of Computational Physics*, 79(1):50–69.
- Foisy, A. and Hayward, V. (1994). A safe swept volume method for collision detection. *International Journal of Robotic Research - IJRR*.
- Fortes, A. F., Joseph, D. D., and Lundgren, T. S. (1987). Nonlinear mechanics of fluidization of beds of spherical particles. *Journal of Fluid Mechanics*, 177:467–483.
- Franciolini, M., Crivellini, A., and Nigro, A. (2017). On the efficiency of a matrix-free linearly implicit time integration strategy for high-order Discontinuous Galerkin solutions of incompressible turbulent flows. *Computers & Fluids*, 159:276–294.
- Garcia-Alonso, A., Serrano, N., and Flaquer, J. (1994). Solving the collision detection problem. *IEEE Computer Graphics and Applications*, 14(3):36–43.
- Gassner, G. J. and Beck, A. D. (2013). On the accuracy of high-order discretizations for underresolved turbulence simulations. *Theoretical and Computational Fluid Dynamics*, 27(3-4):221–237.

- Gilbert, E. G. and Foo, C. P. (1989). Computing the distance between smooth objects in three dimensional space. In *International Conference on Robotics and Automation Proceedings*, volume 6, pages 53–61.
- Glowinski, R. (2003). Finite element methods for incompressible viscous flow. In *Handbook of Numerical Analysis*, volume 9 of *Numerical Methods for Fluids (Part 3)*, pages 3–1176. Elsevier.
- Glowinski, R., Pan, T. W., Hesla, T. I., and Joseph, D. D. (1999). A distributed Lagrange multiplier/fictitious domain method for particulate flows. *International Journal of Multiphase Flow*, 25(5):755–794.
- Glowinski, R., Pan, T. W., Hesla, T. I., Joseph, D. D., and Périaux, J. (2001). A Fictitious Domain Approach to the Direct Numerical Simulation of Incompressible Viscous Flow past Moving Rigid Bodies: Application to Particulate Flow. *Journal of Computational Physics*, 169(2):363–426.
- Grama, A., Karypis, G., Kumar, V., and Gupta, A. (2003). *Introduction to Parallel Computing*. Addison Wesley, 2. edition.
- Gross, D., Hauger, W., Schröder, J., and Wall, W. A., editors (2008). *Kinetik*. Number Bd. 3 in *Technische Mechanik*. Springer, Berlin, 10. edition. OCLC: 254922217.
- Groß, S. and Reusken, A. (2007). An extended pressure finite element space for two-phase incompressible flows with surface tension. *Journal of Computational Physics*, 224(1):40–58.
- Gustafson, J. L. (1988). Reevaluating Amdahl’s Law. *Commun. ACM*, 31(5):532–533.
- Hager, G. and Wellein, G. (2011). *Introduction to High Performance Computing for Scientists and Engineers*. Chapman & Hall/CRC Computational Science Series ; 7. CRC Press, Boca Raton, FL.
- Hamada, K. and Hori, Y. (1996). Octree-based approach to real-time collision-free path planning for robot manipulator. In *Proceedings of 4th IEEE International Workshop on Advanced Motion Control - AMC '96 - MIE*, volume 2, pages 705–710 vol.2.
- Herman, M. (1986). Fast, three-dimensional, collision-free motion planning. In *1986 IEEE International Conference on Robotics and Automation Proceedings*, volume 3, pages 1056–1063.
- Hesthaven, J. S. and Warburton, T. (2008). *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Texts in Applied Mathematics. Springer-Verlag, New York.
- Höfler, K. and Schwarzer, S. (2000). Navier-Stokes simulation with constraint forces: Finite-difference method for particle-laden flows and complex geometries. *Physical Review E*, 61(6):7146–7160.
- Hou, G., Wang, J., and Layton, A. (2012). Numerical Methods for Fluid-Structure Interaction — A Review. *Communications in Computational Physics*, 12(2):337–377.

- Hu, H. H. (1996). Direct simulation of flows of solid-liquid mixtures. *International Journal of Multiphase Flow*, 22(2):335–352.
- Hu, H. H., Joseph, D. D., and Crochet, M. J. (1991). Direct simulation of fluid particle motions. *Theoretical and Computational Fluid Dynamics*, 3(5):285–306.
- Iwainsky, C., Altenfeld, R., an Mey, D., and Bischof, C. (2012). Enhancing Brainware Productivity through a Performance Tuning Workflow. In *Euro-Par 2011: Parallel Processing Workshops*, Lecture Notes in Computer Science, pages 198–207. Springer Berlin Heidelberg.
- Jayanti, S. (2018). *Computational Fluid Dynamics for Engineers and Scientists*. Springer Netherlands.
- Jiménez, P., Thomas, F., and Torras, C. (2001). 3D collision detection: A survey. *Computers & Graphics*, 25(2):269–285.
- Jimenez, P. and Torras, C. (1995). Collision detection: A geometric approach. In *Modelling and Planning for Sensor Based Intelligent Robot Systems*, volume 21 of *Series in Machine Perception and Artificial Intelligence*, pages 68–85. World Scientific.
- Kajishima, T. and Takiguchi, S. (2002). Interaction between particle clusters and particle-induced turbulence. *International Journal of Heat and Fluid Flow*, 23(5):639–646.
- Kajishima, T., Takiguchi, S., Hamasaki, H., and Miyake, Y. (2001). Turbulence Structure of Particle-Laden Flow in a Vertical Plane Channel Due to Vortex Shedding. *JSME International Journal Series B Fluids and Thermal Engineering*, 44(4):526–535.
- Karypis, G. and Kumar, V. (1998). A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392.
- Kay, D., Loghin, D., and Wathen, A. (2002). A Preconditioner for the Steady-State Navier–Stokes Equations. *SIAM Journal on Scientific Computing*, 24(1):237–256.
- Kelley, C. (2003). *Solving Nonlinear Equations with Newton’s Method*. Fundamentals of Algorithms. Society for Industrial and Applied Mathematics.
- Klein, B., Kummer, F., Keil, M., and Oberlack, M. (2015). An extension of the SIMPLE based discontinuous Galerkin solver to unsteady incompressible flows. *International Journal for Numerical Methods in Fluids*, 77(10):571–589.
- Klein, B., Kummer, F., and Oberlack, M. (2012). A SIMPLE based discontinuous Galerkin solver for steady incompressible flows. *Journal of Computational Physics*, pages 235–250.
- Klein, B., Müller, B., Kummer, F., and Oberlack, M. (2016). A high-order discontinuous Galerkin solver for low Mach number flows. *International Journal for Numerical Methods in Fluids*, 81(8):489–520.
- Klosowski, J. T., Held, M., Mitchell, J. S. B., Sowizral, H., and Zikan, K. (1998). Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Trans. Vis. Comput. Graph.*, 4:21–36.

- Krause, D. and Kummer, F. (2017). An incompressible immersed boundary solver for moving body flows using a cut cell discontinuous Galerkin method. *Computers & Fluids*, 153:118–129.
- Kummer, F. (2012). *The BoSSS Discontinuous Galerkin Solver for Incompressible Fluid Dynamics and an Extension to Singular Equations*. PhD thesis, Technische Universität Darmstadt, Darmstadt.
- Kummer, F. (2016). Extended discontinuous Galerkin methods for two-phase flows: The spatial discretization. *International Journal for Numerical Methods in Engineering*, 109(2):259–289.
- Kummer, F., Müller, B., and Utz, T. (2017). Time integration for extended discontinuous Galerkin methods with moving domains. *International Journal for Numerical Methods in Engineering*, 113(5):767–788.
- Lai, M.-C. and Peskin, C. S. (2000). An Immersed Boundary Method with Formal Second-Order Accuracy and Reduced Numerical Viscosity. *Journal of Computational Physics*, 160(2):705–719.
- Lax, P. D. (1954). Weak solutions of nonlinear hyperbolic equations and their numerical computation. *Communications on Pure and Applied Mathematics*, 7(1):159–193.
- Leon, S. J., Björck, Å., and Gander, W. (2013). Gram-Schmidt orthogonalization: 100 years and more. *Numerical Linear Algebra with Applications*, 20(3):492–532.
- Liao, C.-C., Chang, Y.-W., Lin, C.-A., and McDonough, J. M. (2010). Simulating flows with moving rigid boundary using immersed-boundary method. *Computers & Fluids*, 39(1):152–167.
- Lu, X. Y. and Dalton, C. (1996). Calculation of the timing of vortex formation from an oscillating cylinder. *Journal of Fluids and Structures*, 10(5):527–541.
- Luo, H., Dai, H., and Ferreira de Sousa, P. (2009). A hybrid formulation to suppress the numerical oscillations caused by immersed moving boundaries. In *APS Division of Fluid Dynamics Meeting Abstracts*.
- Luo, H., Yin, B., Dai, H., and Doyle, J. (2010). A 3D Computational Study of the Flow-Structure Interaction in Flapping Flight. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics.
- Maury, B. (1996). Characteristics ALE Method for the Unsteady 3D Navier-Stokes Equations with a Free Surface. *International Journal of Computational Fluid Dynamics*, 6(3):175–188.
- Maury, B. (1997). A many-body lubrication model. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 325(9):1053–1058.
- Maury, B. (1999). Direct Simulations of 2D Fluid-Particle Flows in Biperiodic Domains. *Journal of Computational Physics*, 156(2):325–351.

- Maxey, M. (2017). Simulation Methods for Particulate Flows and Concentrated Suspensions. *Annual Review of Fluid Mechanics*, 49(1):171–193.
- Meneghini, J. R. and Bearman, P. W. (1995). Numerical Simulation of High Amplitude Oscillatory Flow About a Circular Cylinder. *Journal of Fluids and Structures*, 9(4):435–455.
- Mittal, R. (1999). A Fourier–Chebyshev spectral collocation method for simulating flow past spheres and spheroids. *International Journal for Numerical Methods in Fluids*, 30(7):921–937.
- Mittal, R., Dong, H., Bozkurttas, M., Najjar, F. M., Vargas, A., and von Loebbecke, A. (2008). A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. *Journal of Computational Physics*, 227(10):4825–4852.
- Mittal, R. and Iaccarino, G. (2005). Immersed Boundary Methods. *Annual Review of Fluid Mechanics*, 37(1):239–261.
- Moës, N., Dolbow, J., and Belytschko, T. (1999). A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46(1):131–150.
- Moore, G. (1965). Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117.
- Moore, M. and Wilhelms, J. (1988). Collision Detection and Response for Computer Animation. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '88*, pages 289–298, New York, NY, USA. ACM.
- Morsi, S. A. and Alexander, A. J. (1972). An investigation of particle trajectories in two-phase flow systems. *Journal of Fluid Mechanics*, 55(2):193–208.
- MPI Forum (2015). *MPI: A Message-Passing Interface Standard, Version 3.1*. University of Tennessee, Knoxville, Tennessee.
- Müller, B. (2014). *Methods for higher order numerical simulations of complex inviscid fluids with immersed boundaries*. Dissertation, Technische Universität Darmstadt, Darmstadt.
- Müller, B., Krämer-Eis, S., Kummer, F., and Oberlack, M. (2016). A high-order Discontinuous Galerkin method for compressible flows with immersed boundaries. *International Journal for Numerical Methods in Engineering*, 110:3–30.
- Müller, B., Kummer, F., and Oberlack, M. (2013). Highly accurate surface and volume integration on implicit domains by means of moment-fitting. *International Journal for Numerical Methods in Engineering*, 96(8):512–528.
- Ofenbeck, G., Steinmann, R., Caparros, V., Spampinato, D. G., and Püschel, M. (2014). Applying the roofline model. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 76–85.
- OpenMP Board (2015). *OpenMP Application Programming Interface, Version 4.5*. OpenMP Architecture Review Board.

- Osterhage, W. W. (2016). Computer-Performance allgemein. In Osterhage, W. W., editor, *Mathematische Algorithmen und Computer-Performance kompakt*, IT kompakt, pages 3–30. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Patankar, N. A., Singh, P., Joseph, D. D., Glowinski, R., and Pan, T. W. (2000). A new formulation of the distributed Lagrange multiplier/fictitious domain method for particulate flows. *International Journal of Multiphase Flow*, 26(9):1509–1524.
- Patankar, S. V. and Spalding, D. B. (1972). A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal of Heat and Mass Transfer*, 15(10):1787–1806.
- Peskin, C. S. (1972). *Flow Patterns around Heart Valves : A Digital Computer Method for Solving the Equations of Motion*. Dissertation, Yeshiva University, New York.
- Peskin, C. S. (2002). The immersed boundary method. *Acta Numerica*, 11:479–517.
- Rabenseifner, R. (2015). *Parallel Programming Workshop*. Course Material for HLRS Course 2015-PAR. HLRS, Universität Stuttgart, Lehrstuhl für Höchstleistungsrechnen, 160. edition.
- Rabenseifner, R. and Wellein, G. (2005). Comparison of Parallel Programming Models on Clusters of SMP Nodes. In Bock, H. G., Phu, H. X., Kostina, E., and Rannacher, R., editors, *Modeling, Simulation and Optimization of Complex Processes*, pages 409–425. Springer Berlin Heidelberg.
- Reed, W. H. and Hill, T. R. (1973). Triangular mesh methods for the neutron transport equation. Technical Report LA-UR-73-479; CONF-730414-2, Los Alamos Scientific Lab., N.Mex. (USA).
- Saad, Y. and Schultz, M. (1986). GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869.
- Saviankou, P. and Cube developer community (2018a). CubeLib: General purpose C++ library and tools. Zenodo.
- Saviankou, P. and Cube developer community (2018b). CubeW: High performance C Writer library. Zenodo.
- Saviankou, P., Visser, A., and Cube developer community (2018). CubeGUI: Graphical explorer. Zenodo.
- Saye, R. (2016). Interfacial gauge methods for incompressible fluid dynamics. *Science Advances*, 2(6):e1501869.
- Schäfer, M., Turek, S., Durst, F., Krause, E., and Rannacher, R. (1996). Benchmark Computations of Laminar Flow Around a Cylinder. In Hirschel, P. D. E. H., editor, *Flow Simulation with High-Performance Computers II*, number 48 in Notes on Numerical Fluid Mechanics (NNFM), pages 547–566. Vieweg+Teubner Verlag.

- Schenk, O. and Gärtner, K. (2002). Two-level dynamic scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems. *Parallel Computing*, 28(2):187–197.
- Schenk, O. and Gärtner, K. (2004). Solving unsymmetric sparse systems of linear equations with PARDISO. *Future Generation Computer Systems*, 20(3):475–487.
- Schenk, O. and Gärtner, K. (2006). On fast factorization pivoting methods for sparse symmetric indefinite systems. *Electronic Transactions on Numerical Analysis*, 23(1):158–179.
- Seo, J. H. and Mittal, R. (2011). A sharp-interface immersed boundary method with improved mass conservation and reduced spurious pressure oscillations. *Journal of Computational Physics*, 230(19):7347–7363.
- Shahbazi, K., Fischer, P. F., and Ethier, C. R. (2007). A high-order discontinuous Galerkin method for the unsteady incompressible Navier–Stokes equations. *Journal of Computational Physics*, 222(1):391–407.
- Sigurgeirsson, H., Stuart, A., and Wan, W.-L. (2001). Algorithms for Particle-Field Simulations with Collisions. *Journal of Computational Physics*, 172(2):766–807.
- Silvester, D., Elman, H., Kay, D., and Wathen, A. (2001). Efficient preconditioning of the linearized Navier–Stokes equations for incompressible flow. *Journal of Computational and Applied Mathematics*, 128(1):261–279.
- Sipser, M. (2006). *Introduction to the Theory of Computation*. Thomson Course Technology, Boston, 2nd edition.
- Smith, B. F. (1997). Domain Decomposition Methods for Partial Differential Equations. In *Parallel Numerical Algorithms*, ICASE/LaRC Interdisciplinary Series in Science and Engineering, pages 225–243. Springer, Dordrecht.
- Sun, X. H. and Ni, L. M. (1993). Scalable Problems and Memory-Bounded Speedup. *Journal of Parallel and Distributed Computing*, 19(1):27–37.
- Tornero, J., Hamlin, J., and Kelley, R. B. (1991). Spherical-object representation and fast distance computation for robotic applications. In *1991 IEEE International Conference on Robotics and Automation Proceedings*, pages 1602–1608 vol.2.
- Uhlmann, M. (2005). An Immersed Boundary Method with Direct Forcing for the Simulation of Particulate Flows. *Journal of Computational Physics*, 209(2):448–476.
- Utz, T. and Kummer, F. (2018). A high-order discontinuous Galerkin method for extension problems. *International Journal for Numerical Methods in Fluids*, 86(8):509–518.
- Utz, T., Kummer, F., and Oberlack, M. (2017). Interface-preserving level-set reinitialization for DG-FEM. *International Journal for Numerical Methods in Fluids*, 84(4):183–198.
- van den Bergen, G. (2003). *Collision Detection in Interactive 3D Environments*. CRC Press.

- Vanekckek, G. (1994). Back-face culling applied to collision detection of polyhedra. *Journal of Visualization and Computer Animation*, 5:55–63.
- Wan, D. and Turek, S. (2006). Direct numerical simulation of particulate flow via multigrid FEM techniques and the fictitious boundary method. *International Journal for Numerical Methods in Fluids*, 51(5):531–566.
- Wan, D. and Turek, S. (2007). An efficient multigrid-FEM method for the simulation of solid–liquid two phase flows. *Journal of Computational and Applied Mathematics*, 203(2):561–580.

A Appendix

A.1 A localized operator preconditioner

The intention of this method is strongly related to the Schur preconditioning ansatz. Basically, the idea is to yield a cell local approximation to the inverse of the convection-diffusion operator as it occurs in (6.12) within M_S . Therefore, the cell local convection-diffusion operator multiplied by a testfunction \vec{v} and integrated over a cell K_j reads

$$\int_{K_j} \nabla \cdot (\rho_f(\vec{w} \otimes \vec{u}) + \mu_f \nabla \vec{u}) \cdot \vec{v} dV. \quad (\text{A.1})$$

Assuming the velocity field to be local divergence free ($\nabla \cdot (\vec{w} \otimes \vec{u}) = \vec{w} \cdot \nabla \vec{u}$) and applying integration by parts to the diffusive part yields

$$\rho_f \int_{K_j} (\vec{w} \cdot \nabla \vec{u}) \cdot \vec{v} dV = \mu_f \int_{\partial K_j} (\nabla \vec{u} \cdot \vec{n}_{K_j}) \cdot \vec{v} dS - \mu_f \int_{K_j} \nabla \vec{u} \cdot \nabla \vec{v} dS. \quad (\text{A.2})$$

Now, the diffusive flux over the cell boundary is locally described by a flux into cell K_j and out of K_j :

$$\mu_f \int_{\partial K_j} (\nabla \vec{u} \cdot \vec{n}_{K_j}) \cdot \vec{v} dS = \mu_f \int_{\partial K_j} (\nabla \vec{u}_{\text{in}} \cdot \vec{n}_{\partial K_j} \cdot \vec{v}_{\text{in}} - \nabla \vec{u}_{\text{out}} \cdot \vec{n}_{\partial K_j} \cdot \vec{v}_{\text{out}}) dS. \quad (\text{A.3})$$

The main intention is that this local convection-diffusion matrix can be inverted much easier than the global one. Additionally, the local matrix is supposed to be a sufficient approximation to the full convection-diffusion matrix and is therefore suitable for efficient preconditioning.

Curriculum vitae

Der Lebenslauf ist aus Datenschutzgründen in der Online-Version nicht enthalten.