Technische Universität München
Lehrstuhl für Informatik mit Schwerpunkt
Wissenschaftliches Rechnen

Hybrid Multiscale Simulation Approaches for
Micro- and Nanoflows

Philipp Rainer Neumann

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des Akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. H. Räcke
Prüfer der Dissertation: 1. Univ.-Prof. Dr. H.-J. Bungartz
2. Priv.-Doz. Dr. J. Harting,
   Technische Universität Eindhoven/ Niederlande
   (schriftliche Beurteilung)
3. Univ.-Prof. Dr. U. Rüde,
   Friedrich-Alexander-Universität Erlangen-Nürnberg

Die Dissertation wurde am 17.04.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 25.06.2013 angenommen.
Contents

Abstract 5
Zusammenfassung 7
Acknowledgement 9
List of symbols 11

I Introduction 13
1 Scales in Physics and Impressionism 13
2 On Multiscale Problems in Computational Fluid Dynamics 14
2.1 Multiscale Modelling 14
2.2 Multiscale Scenarios and Scale Transitions 17
2.3 Software Challenges 19
2.4 State-of-the-Art Software for Multiscale Fluid Dynamics Applications 20
3 Aims and Outline 24

II Theoretical Foundation: Modelling Flows on Different Scales 27
4 Microscopic Modelling: Molecular Dynamics 27
4.1 Model Description 27
4.2 Discretisation and Numerical Schemes 28
4.3 Features and Limitations 29
5 Mesoscopic Modelling: The Lattice Boltzmann Method 30
5.1 Introduction 30
5.2 Model Description 30
5.3 The Collision Operator: Assumptions and Models 32
5.4 Adaptive Formulations 34
5.5 Boundary Conditions 36
5.6 Moving Obstacles 39
5.7 Extension to Finite Knudsen Numbers 42
5.8 Features and Limitations 45
6 Macroscopic Modelling: The Navier-Stokes Equations 46
6.1 Model Description 46
6.2 Discretisation and Numerical Schemes 48
6.3 Particle-in-Flow Simulations Based on Faxén’s Theorems 49
6.4 Features and Limitations 51
7 From Micro to Macro: Scale Transitions 51
7.1 From Molecules to Populations 52
7.2 From Populations to Macroscopic Conservation Laws 55

III Algorithms and Implementations 59
8 Molecular Dynamics Simulations 59
8.1 MarDyn 60
8.2 Built-in Molecular Dynamics 60
9 A Short Introduction to Peano 62
## Contents

10 Lattice Boltzmann Implementations within Peano ........................................... 66
  10.1 Motivation .................................................................................. 66
  10.2 Efficiency and Compatibility ......................................................... 67
  10.3 The Block-Structured Lattice Boltzmann Application in Peano ........... 69
  10.4 Implementation of the Lattice Boltzmann Algorithm ......................... 71
    10.4.1 Collision Models .................................................................. 72
    10.4.2 Boundary Treatment ........................................................... 73
    10.4.3 Streaming .......................................................................... 73
  10.5 Implementations on the Spacetree Grid ............................................ 74
    10.5.1 Static Adaptivity .................................................................. 76
    10.5.2 Dynamic Grid Refinement ...................................................... 79
11 Navier-Stokes Simulations within Peano ....................................................... 82
12 Lattice Boltzmann–Navier-Stokes Simulations ............................................ 84
  12.1 Motivation .................................................................................. 84
  12.2 Exchange of Flow Quantities: An Optimisation Approach ................. 85
  12.3 Coupled Implementations within Peano ........................................... 89
    12.3.1 Technical Details ................................................................. 89
    12.3.2 Two-Way Flow Coupling ....................................................... 91
    12.3.3 A Hierarchical Hybrid Particle Transport Method .................... 92
13 Molecular Dynamics–Lattice Boltzmann Simulations ................................... 96
  13.1 Motivation .................................................................................. 96
  13.3 Prototype Implementations: Identification of Software Requirements .... 105
  13.4 The Macro-Micro-Coupling Tool (MaMiCo) ....................................... 108

IV Applications and Numerical Results ............................................................. 115

14 Lattice Boltzmann Simulations within Peano: Validation and Performance ........ 116
15 Flows at Finite Knudsen Numbers: Validation and Microreactor Simulations .... 125
16 Simulation of Particle Translocation in Drift Ratchets ................................. 132
  16.1 Short-Time Particle Displacement ............................................... 133
  16.2 Long-Time Simulations: Lattice Boltzmann–Navier-Stokes Methods ..... 139
17 Molecular Dynamics–Lattice Boltzmann Coupling ..................................... 146
  17.1 MaMiCo: Validation and Performance .......................................... 146
  17.2 On RDF-Based Boundary Forcing .................................................. 153
  17.3 Towards the Molecular–Continuum Simulation of Nanofilters ............ 158

V Summary and Outlook ..................................................................................... 165

Appendix ............................................................................................................. 169

Bibliography ....................................................................................................... 173
Abstract

The simulation of flows over a wide range of spatial or temporal scales has turned out to be one of the most challenging and important fields in computational fluid dynamics. In order to study flow phenomena whose characteristics evolve on different scales or in the transition regime between the continuum, the statistical or the molecular scale, coupled multiscale methods are required. These hybrid methods represent a compromise between physical accuracy and computational complexity. Examples comprise molecular dynamics–Lattice Boltzmann simulations for nanoflows or hybrid continuum–statistical methods for rarefied gas flows where parts of the respective domains are solved by either coarse- or fine-scale simulation methods.

For the development of these scale-coupling algorithms, accurate mathematical and physical models of the scale transition regime are required. Efficient sequential and parallel implementations of the single-scale components are necessary to solve the underlying flow problem in reasonable time. Besides, a well-fitting software environment needs to be chosen for the development of the single-scale solvers. One particular environment is given by Peano, a framework for spatially adaptive mesh-based simulations. Peano already contains a sophisticated Navier-Stokes solver for the study of continuum phenomena. Fine-scale simulation components—such as Lattice Boltzmann or molecular dynamics solvers—and respective coupled simulations, however, have not been integrated in the framework yet. Finally, the simulation software for the coupled multiscale system needs to provide a flexible and modular environment for the further development of new coupling strategies as well as an efficient and parallel treatment of the different coupling steps.

In this thesis, a spatially adaptive Lattice Boltzmann scheme is incorporated into Peano and extends the applicability of the framework from the continuum to the statistical scale. A modular development of coupled algorithms is guaranteed via the design principles of Peano. The software is validated in benchmark computations and applied to micro- and nanoflow scenarios such as rarefied gas flows in microreactors or particle transport in nanopores. For the latter, an adaptive mesh refinement technique has been established which allows for the dynamic spatial refinement of particular flow regions. Besides, a new hybrid Lattice Boltzmann–Navier-Stokes method is presented and applied to the particle transport scenarios. In order to go beyond the statistical scale, a coupling tool for massively parallel molecular dynamics–Lattice Boltzmann simulations has been developed. Based on the analysis of existing coupling schemes, it encapsulates all coupling steps in different modules; this reduces the efforts in setting up new coupling schemes to the exchange of one or several available module implementations. To the author’s knowledge, the coupling tool hence provides the first piece of software for molecular dynamics–Lattice Boltzmann simulations with this high level of modularity on the one hand and applicability to massively parallel scenarios on the other hand. The capabilities of the tool are demonstrated in different molecular dynamics–Lattice Boltzmann scenarios.


I want to thank Prof. Dr. Hans-Joachim Bungartz for his continuous support throughout the last four years. I enjoyed the opportunities at SCCS a lot, including (but definitely not limited to) my research stays in Eindhoven, the freedom to branch out into various directions of computational fluid dynamics and of course the table football :-) A particular thanks also goes to Prof. Dr. Ulrich Rüde and the waLBerla crew who woke my interest in meso- and multiscale flow simulations and the nice times I spent with them during my diploma thesis as well as during the last years when I found my way back to Erlangen from time to time. I further thank Prof. Dr. Jens Harting for all his suggestions with respect to molecular–continuum coupling schemes, the related challenges, underlying physics and so forth—without your support, I think I would have hardly made it.

I also want to thank all students who contributed to this work in the context of student jobs, master and bachelor theses. In particular, I want to say “Thank you!” to Till Rohrmann for his work on finite Knudsen flows, Nikola Tchipev for his contribution to the parallelisation of the built-in molecular dynamics solver of the macro-micro-coupling tool and Denis Jarema for his great work on hybrid Lattice Boltzmann–Navier-Stokes-based particle transport simulations.

Together with Atanas Atanasov, I could investigate and test different parallelisation concepts within Peano. Thank you very much, Atanas, for all your work and our fruitful discussions on this topic!

Many people helped me to improve this thesis. I want to thank Dr. Tobias Weinzierl, Dr. Tobias Neckel, Wolfgang Eckhardt, Simon Bogner, Gerrit Buse and my girlfriend Julia for all their remarks and feedback.

I especially want to thank Julia for all her patience during the last four years when I showed her a multitude of spheres moving forth and back or weird arrows which travel from one cell into another. It’s more than nine amazing years with you now and I hope we’ll have another 80 of those in future.

Finally, I want to thank the Münchner Verkehrsgesellschaft for providing comfortable seats and regular delays which I found to be extremely useful to write the one or the other section of this thesis.
List of Abbreviations and Symbols

Symbols

c_i \quad \text{lattice velocity}
c_s \quad \text{speed of sound}
F \quad \text{force}
D \quad \text{spatial dimension}
dt \quad \text{time step}
dx \quad \text{mesh size}
f_i(x, t) \quad \text{particle distribution function belonging to lattice velocity } c_i
f_i^{eq} \quad \text{equilibrium distribution}
f_i^{neq} \quad \text{non-equilibrium part of a particle distribution, i.e. } f_i = f_i^{eq} + f_i^{neq}
g \quad \text{continuous particle distribution}
g_i^{eq} \quad \text{Maxwellian distribution}
p \quad \text{momentum}
k_B \quad \text{Boltzmann’s constant}
Kn \quad \text{Knudsen number}
l \quad \text{characteristic length scale in fluid dynamic problems}
L_p \quad \text{angular momentum}
m_p \quad \text{mass of a molecule/ particle}
M \quad \text{transformation matrix to moment space}
Ma \quad \text{Mach number}
n \quad \text{number density}
p(x, t) \quad \text{pressure}
Q \quad \text{number of lattice velocities}
R_p \quad \text{orientation of a particle}
Re \quad \text{Reynolds number}
\Delta s \quad \text{step size for the molecule displacement in the USHER scheme}
t \quad \text{time}
u(x, t) \quad \text{(macroscopic) velocity vector}
U \quad \text{potential energy}
v_p \quad \text{velocity of a particle } p
w_i \quad \text{lattice weight}
x \quad \text{position vector in space}

\Gamma \quad \text{relaxation matrix for the hydrodynamic and ghost modes}
\delta_{\alpha\beta} \quad \text{Kronecker delta}
\delta(s) \quad \text{Dirac delta function}
\epsilon \quad \text{depth of Lennard-Jones potential}
\epsilon_{\kappa_n} \quad \text{expansion parameter in Chapman-Enskog expansion}
\lambda \quad \text{mean free path}
\mu \quad \text{dynamic viscosity}
\nu \quad \text{kinematic viscosity}
\nu_{bulk} \quad \text{bulk viscosity}
**LIST OF ABBREVIATIONS AND SYMBOLS**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Pi_{\alpha \beta} )</td>
<td>stress tensor</td>
</tr>
<tr>
<td>( \rho(x, t) )</td>
<td>density</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>characteristic length of Lennard-Jones potential</td>
</tr>
<tr>
<td>( \tau )</td>
<td>relaxation time</td>
</tr>
<tr>
<td>( \tau_{\alpha \beta} )</td>
<td>entries of the (Newtonian) stress tensor</td>
</tr>
<tr>
<td>( \tau_p(t) )</td>
<td>torque</td>
</tr>
<tr>
<td>( \phi )</td>
<td>matrix controlling the thermal noise in fluctuating LB simulations</td>
</tr>
<tr>
<td>( \omega )</td>
<td>angular velocity of a particle</td>
</tr>
<tr>
<td>( \mathbb{I} )</td>
<td>identity matrix</td>
</tr>
<tr>
<td>( \Delta_i (f - f^{eq}) )</td>
<td>collision operator applied to ( f - f^{eq} )</td>
</tr>
<tr>
<td>( \nabla )</td>
<td>gradient with respect to the spatial coordinates ( x )</td>
</tr>
<tr>
<td>( \nabla_{\mathbf{v}} )</td>
<td>gradient with respect to the velocity coordinates ( \mathbf{v} )</td>
</tr>
<tr>
<td>( \partial (k) )</td>
<td>( (k\text{-th}) ) partial derivative with respect to time</td>
</tr>
<tr>
<td>( \partial_{x_{\alpha}} (k) )</td>
<td>( (k\text{-th}) ) partial derivative with respect to spatial coordinate ( x_{\alpha} )</td>
</tr>
<tr>
<td>( \langle \cdot \rangle )</td>
<td>Average of ( \cdot )</td>
</tr>
<tr>
<td>( \frac{d}{dt} )</td>
<td>(total) derivative with respect to time</td>
</tr>
</tbody>
</table>

### Sub- and Superscripts

- \( t_C \) convective time scale
- \( t_D \) diffusive time scale
- \( t_H \) hydrodynamic length scale
- \( f_i^* \) post-collision state of the distribution \( f_i \)
- \( i \) index of inverse lattice velocity vector to \( c_i \), i.e. \( c_i = -c_i \)
- \( s^{LB} \) quantity \( s \) obtained from a Lattice Boltzmann simulation
- \( s^{MD} \) quantity \( s \) obtained from a molecular dynamics simulation
- \( s^{NS} \) quantity \( s \) obtained from a Navier-Stokes simulation
- \( s^\top \) transpose of matrix \( s \)

### Abbreviations

- AMR adaptive mesh refinement
- BGK Bhatnagar-Gross-Krook [13]
- \( dQ_d \) \( d \)-dimensional velocity discretisation model with \( q \) lattice velocities
- DSMC Direct Simulation Monte Carlo
- FLB fluctuating Lattice Boltzmann
- LB Lattice Boltzmann
- LJ Lennard-Jones
- MD molecular dynamics
- MRT multiple-relaxation-time
- NS Navier-Stokes
- RDF radial distribution function
Part I

Introduction

1 Scales in Physics and Impressionism

Visiting the Musée Marmottan Monet in Paris, you may look at more than three hundred works by Claude Monet, one of the first Impressionist painters. Amongst others, his masterpiece “Impression, soleil levant” which gave rise to the name Impressionism is shown in this museum as well, cf. Fig. 1.1. Considering Monet’s painting, different levels of detail can be observed. Zooming into the lower right part of the painting, some dark horizontal stripes become visible. Zooming out, these stripes are recognised to resemble waves which form (more or less) regular structures in the sea. Similar to zooming in and out of Monet’s ocean, different levels of detail can be observed in fluid dynamics and other engineering disciplines as well: on the one hand, there are molecules whizzing around in a random manner, changing their movements within femtoseconds. On the other hand, one may consider the flow around a car which drives through a tunnel. Considering these two cases, the spatial and temporal scales between the molecular and the automotive studies differ by a factor of $10^{15}$ or—written in a different form—1 000 000 000 000 000; this is similar to comparing the distance between earth and sun with the size of a needle eye!

Both scenarios from the field of fluid dynamics have a strong impact on our daily life. Microscopic descriptions of fluids such as molecular models are for example required for the development and understanding of new biotechnological, pharmaceutical or environmental devices. However, experimental measurements are often not possible due to the very tiny temporal and length scales involved. Besides, the engineer could not optimise or design a car without having a valid macroscopic description of the flow at hand—molecular investigations would require zillions of measurements! Numerical simulations of respective flow problems in automotive engineering would thus be impossible on the molecular scale: considering state-of-the-art numerical methods, the biggest molecular dynamics simulations that have run on supercomputers handled trillions, that is $O(10^{12})$, molecules [46, 64]. For an ideal gas, this would correspond to a volume of $3.7 \cdot 10^{-11}$ liters at standard conditions. It immediately becomes clear that the pure consideration of the smallest scale is not a general option.

Sticking to the example of simulating the flow around a car, molecular investigations would

---

be very difficult to handle also with respect to the amount of simulation output: huge chunks of information are available via the simulation of molecular systems, although an engineer might only be interested in averaged quantities such as the mean flow velocity or the pressure distribution close to the car. Still, even when using a macroscopic model to solve the flow problem, molecular effects must not be completely neglected, but need to be adequately included. For example, the viscosity of the fluid which evolves from the random movement of the molecules and describes diffusive effects is of major importance in most respective flow scenarios. Hence, both micro- and macroscopic approaches and an adequate description of their relations and interactions are necessary to solve and investigate flow problems on different scales.

One may even go one step further: for particular flow problems, it is a priori not clear which scale fits best to describe the flow. For example, in the case of simulating micro- and nanoflows [98], microscopic methods such as molecular dynamics are required on the one hand to capture physics on the molecular scale correctly. On the other hand, for complex devices from micro- and nanotechnology, the computational costs of the arising simulation might already be much too big. As a consequence, a single-scale description of the problem is not sufficient anymore: different scales need to be considered and combined in a multiscale approach to investigate the complex problem.

In this thesis, I focus on the numerical simulation of flow problems by considering flow modelling on three different scales: the micro- and macroscale which have already been addressed above and the mesoscale which allows for a statistical description of microscopic effects. It is hence located between the micro- and the macroscopic scale. I describe different models (Part II) and respective implementations (Part III) for each scale under consideration. In particular, I also provide theoretical considerations and implementational techniques to connect these models which allows for efficient multiscale fluid simulations. Numerical results for various single- and multiscale scenarios are subject of Part IV.

In the following chapter, different aspects on multiscale fluid dynamics such as multiscale modelling (Sec. 2.1), different respective flow scenarios (Sec. 2.2) and software challenges (Sec. 2.3) are considered. Besides, available state-of-the-art simulation software for multiscale fluid dynamics is discussed in Sec. 2.4. Part I closes with a short overview of the contents of this thesis.

2 On Multiscale Problems in Computational Fluid Dynamics

2.1 Multiscale Modelling

As mentioned before, flow processes may occur on different scales. Most points of the following discussion on modelling different scales are encountered in all engineering disciplines. I focus on the example of fluid dynamics, since this is the particular field of interest in this thesis.

To visualise the separation of different scales, scale separation maps can be used (see for example [84]). A respective illustration of the flow problems that are considered in this thesis and their underlying flow models is shown in Fig. 2.1: the x-axis shows the characteristic time whereas the y-axis represents the characteristic length scale of the respective problems. On the microscopic, i.e. finest, scale under consideration, the fluid can be described by modelling each molecule separately. The single molecules move freely in space and interact via intermolecular forces. Models for flow scenarios on the molecular scale comprise amongst others Newton-based molecular dynamics (MD) approaches or stochastic Monte Carlo models [148]. On the mesoscopic scale, the molecular behaviour is considered in a statistical sense implying the evaluation of particle distributions. Considerations on the mesoscale hence allow for coarse-graining of molecular information and abstain from resolving single molecules. Representative models on the mesoscale are given by the Lattice Boltzmann (LB) method [163], dissipative particle dynamics [86] or stochastic rotation dynamics [120]. On the macroscopic level, the fluid is assumed to obey the laws of continuum
mechanics, i.e. the fluid is considered to homogeneously fill the entire space. Only average flow quantities such as pressure, temperature or flow velocity can be evaluated for each fluid control volume under consideration. The most common description of continuum flows is provided by the Navier-Stokes (NS) equations [162]. The different flow problems—given by red-coloured ellipses in Fig. 2.1—are explained in more depth in the next section. From the map, several aspects of mathematical and physical modelling become apparent. First, the range of application of each model can be placed at a particular position in the scale separation map; the same holds for the (flow) problems that need to be solved. Second, given a certain problem, an accurate model needs to be chosen, based on the location of the problem within the scale separation map and the ranges of application of the different models that are available.

The ranges of application of the models typically overlap and thus yield non-unique modelling possibilities in certain regions of the scale separation map. The models whose range of application comprise the problem under consideration, however, may differ in their mathematical or physical description. Besides, a diversity of numerical and algorithmic realisations can be available to approximate each model. The arising numerical schemes vary in their implementational complexity or in their computational costs. Higher computational costs are typically encountered in the numerical treatment of the problem under consideration by fine-scale models, that is models which contain the problem close to the upper/ right end of their range of application. For example, consider the problem of “(Gas) Flows in Micro-Devices” in Fig. 2.1: parts of the problem are still covered by the region of molecular modelling. In order to solve the problem using a numerical realisation of the fine-scale molecular model, finer spatial resolutions or finer time steps are required in comparison to applicable models from the mesoscopic range of application; this results in higher computational runtimes. Much research is also spent on the extension of existing coarse-scale schemes to finer scales. An example for this is discussed later in Sec. 5.7 where a mesoscopic Lattice Boltzmann scheme is extended to the rarefied gas regime\(^2\) to further increase the range of application for the (green-coloured) mesoscopic model in Fig. 2.1 towards the molecular regime. This, however is not possible for all problems: for example, if there is no coarse-grained model available for particular molecular effects, molecular modelling is unavoidable.

---

\(^2\)Standard Lattice Boltzmann schemes are valid for thermodynamics systems that are close to equilibrium state whereas rarefied gas flows imply local non-equilibrium processes.
A multiscale scheme or multiscale method is a combination of two or several numerical methods which may be based on different mathematical models and describe a given problem on different length or time scales.

Although in literature, the expressions “adaptive” and “multiscale” are sometimes used synonymously, there is a clear separation in this thesis: adaptivity refers to applying the same model on differently resolved domains (in space or time). Multiscale methods refer to methods that involve solvers on different scales and which are (typically) based on different models.

Solving a specific physical problem which appears to span over several scales or—with respect to the scale separation map—is situated near the borders of a certain scale requires answers to several questions:

1. Which scale is/ scales are involved? In case of multiple scales: do all scales need to be considered in the problem description? Or is for example a single coarse-grained problem description available and sufficient in this particular case?

2. Which models are available on each scale involved? Are the model descriptions accurate enough for this specific problem?

3. Which algorithms and numerical methods exist to approximate each mathematical model? Are their computational complexity and the arising computational costs acceptable? Do they yield accurate solutions?

4. In case that more than one scale is involved: how can the different methods be incorporated into a multiscale scheme?
   - How do the numerical schemes need to be coupled? Is for example a uni- or a bidirectional coupling required? Is it computationally efficient to couple the chosen numerical single-scale schemes?
   - Are the overall computational complexity and the computational costs of the multiscale scheme acceptable?
   - Are the results of the multiscale scheme accurate enough?

Answering each of the questions can be highly non-trivial. The choice of the best-suited solution strategy may hence become a very difficult and non-unique task. Fig. 2.2 illustrates the assembling of a multiscale scheme.

Considering the listing of questions from above, the questions on numerical accuracy and computational complexity are repeated in point four on multiscale schemes. This is no coincidence: not every multiscale scheme is more efficient from the computational point of view and consequently faster than a pure fine-scale method. For example, choosing the two computationally most efficient single-scale schemes and combining them into a multiscale method only yields a highly efficient multiscale scheme if the transition between the two

---

[3] Consider the flow modelling from Fig. 2.1 and the transparent clouds at the lower left and upper right; quantum mechanics can be used on sub-molecular scales to describe matter, for example by the Schrödinger equation. On the very large scales, particular models are required as well, amongst others for ocean modelling or weather forecasting.
methods can be established efficiently as well. If the transition is computationally slow, choosing two other single-scale methods and combining them may represent a more promising approach. The choice of a multiscale scheme as illustrated in Fig. 2.2 may hence be considered as an iterative process in this case. Constructing an accurate multiscale scheme can become very challenging; this is particularly the case when the transition between the different scales is not well-understood yet and is still focus of current research. The latter point is picked up again in the context of hybrid molecular dynamics–Lattice Boltzmann (MD–LB, cf. Fig. 2.1) simulations within this thesis.

### 2.2 Multiscale Scenarios and Scale Transitions

With the definition of multiscale schemes at hand, different scenarios from the field of computational fluid dynamics (CFD) are shortly described in the following. The respective scenarios are building blocks for the simulations from Part IV and can be found in the scale separation map from Fig. 2.1. A brief problem description of each scenario is given which demonstrates the limitations of single-scale models and points out the need to correctly model the scale transition, either via extending an existing single-scale method or via hybrid multiscale approaches; for details on the scenarios, the reader is referred to Part IV. Besides, the arising software requirements and challenges are picked up; they are subject to discussion in the subsequent section.

Starting at the upper right part of the scale separation map from Fig. 2.1, the first “overlap of scales” consists in the region between meso- and macroscopic/continuum considerations. A coupling between the two scales can be favourable if major parts of the computational domain are represented by relatively big-sized structures whereas some regions need to be highly resolved and require a particular treatment which is not possible anymore or harder to establish on the purely continuum scales. For example, assume a micro- to nano-sized particle which is suspended in a large reservoir. The overall flow in the reservoir may hardly feel the influence of the particle. The particle, however, is strongly influenced by the motion of the fluid. Besides, due to the very small size of the particle, Brownian fluctuations in the fluid are not negligible anymore. These fluctuations may be incorporated into both meso-
macroscopic models. However, consistently incorporating the fluctuations into continuum
models can be a very challenging task, cf. Sec. 5.8. An alternative approach consists in
modelling major parts of the flow domain by the continuum solver without thermal fluctu-
ations and embedding the particle into a separate region which is resolved by a mesoscopic
method including thermal fluctuation effects. Since the particle is very small, the spatial and
temporal resolution within this mesoscopic domain also needs to be increased compared to
the continuum description of the large reservoir. We thus cross the “border between contin-
uum and mesoscopic flow descriptions” by applying an adaptive high-resolution procedure
to correctly model the flow in this region. A respective simulation scenario on particle trans-
port in nanopores (see Fig. 2.1) is discussed in Chap. 16: a dynamic mesh refinement for
the mesoscopic Lattice Boltzmann model is introduced using different flow models—with
or without considering thermal fluctuations—in differently resolved regions. Afterwards, a
continuum (Navier-Stokes) solver is coupled to the Lattice Boltzmann description to ac-
count for the flow in the large reservoir. Several questions on the hybrid meso-macro-model
arise: how is the coupling between the two flow descriptions established from the modelling
point of view, that is how can the scale transition be modelled consistently? Which pieces
of flow information are extracted from the meso-/ macroscopic method and plugged into
the macro-/ mesoscopic model? For example, is a uni- or bidirectional coupling of the two
models required? How is the coupling established technically within the used piece(s) of
software? How can the adaptivity close to the moving particle be steered? Answers to these
issues are given in Chap. 12 and 16.

Another type of scenarios from the scale separation map in Fig. 2.1 is given by (rarefied) gas
flows in micro-devices. These scenarios are close to the molecular regime: due to the very
small size of micro-devices such as for example micro-electro-mechanical systems (MEMS),
the characteristic scales of the flow problem approach the size of characteristic molecular
quantities. For example, the length scale of micro- to nano-sized devices is of similar order
as the molecular free mean path of air molecules (which is $\approx 68 \text{nm}$ at standard conditions).
As a consequence, coarse-grained continuum or mesoscopic methods hardly capture the cor-
rect flow behaviour. However, the purely molecular regime is not reached yet which makes
molecular descriptions expensive. Hence, several model extensions have been proposed to
account for the molecular effects and thus enlarge the validity range of the coarse-grained
methods towards the molecular regime. In Sec. 5.7 and Chap. 15, the theory and results are
presented which show the capabilities of the mesoscopic Lattice Boltzmann method when
incorporating respective extensions into the underlying model. Within this context, a spa-
tially adaptive scheme is developed which applies the model extensions in fine-scale regions
of the computational domain only. In coarser regions, that is regions of larger characteristic
size, the standard Lattice Boltzmann method is used instead.

To the lower left of the scenario “(Gas) Flows in Micro-Devices”, the scenario “LB–MD Sim-
ulations of Liquids” can be found in the scale separation map, cf. Fig. 2.1. Approaching
the molecular length scales, a molecular modelling of the fluid is required: intermolecular
interactions need to be taken into account, thermal fluctuations within predefined control
volumes occur due to the Brownian movement of the atoms, and also the shape of the single
atoms may have a significant impact on the overall flow behaviour. In case of big simulation
domains, however, the computational costs for a purely molecular simulation of the problem
are not affordable anymore. For this reason, hybrid simulation techniques such as concur-
rent Lattice Boltzmann–molecular dynamics methods are strongly evolving in the field of
nanofluidics and are subject of current research. These methods are based on the idea of
decomposing the computational domain into molecular regions where molecular dynamics
is required to capture the correct flow behaviour and coarse-grained Lattice Boltzmann re-
regions where a mesoscopic consideration of the flow is expected to be sufficient. Since the
mesoscopic simulation is much cheaper as molecular simulations, significantly bigger sce-
narios can be investigated by means of the hybrid simulation at acceptable computational
cost. Several issues arise when creating a respective hybrid simulation model. First, in
contrast to molecular dynamics which belongs to the class of particle methods, the Lattice
Boltzmann method uses Cartesian grids and thus yields a flow description on a fixed grid
in space. Second, while the default Lattice Boltzmann scheme obeys the equation of state
of ideal gases, molecular dynamics simulations typically yield a non-ideal equation of state. This implies different fluid properties at different points in the phase space, for example different regimes for gaseous and liquid state of the fluid or different values for the fluid viscosity. Third, different dimensionless formulations are used on molecular dynamics and Lattice Boltzmann side. A consistent scaling of all relevant quantities such as mass, flow velocities or temperature is thus required. This also comprises the correct representation of the characteristic flow parameters. For example, the fluid in a simple (BGK-based) Lattice Boltzmann simulation is solely characterised via its viscosity. The viscosity parameter, however, is not explicitly given in molecular systems. As a consequence, the viscosity and the molecular interaction model must be matched according to the underlying equations of state. Fourth, since the molecular dynamics region is neighboured to or embedded into the Lattice Boltzmann domain, open boundaries need to be modelled which allow molecules to enter or leave the molecular dynamics region. Classical molecular dynamics simulations use periodic boundary conditions where molecules which leave the domain for example on the right will again enter the domain on the left. This methodology conserves mass and models an infinite domain. For open boundaries, however, the conservation of (thermodynamic) quantities is challenging and requires a physically accurate modelling. Fifth, similar to the macro-to-meso scale coupling that has been mentioned before, a separation in time and length scales occurs in micro-to-meso coupling as well: for example, due to the fine-grain structure of molecular dynamics, tens to hundreds of molecular dynamics time steps need to be performed per Lattice Boltzmann time step. The algorithmic concepts and simulation results for hybrid Lattice Boltzmann–molecular dynamics simulations are discussed in Chap. 13 and 17; cf. Sec. 17.2 for one particular aspect of open boundary modelling.

2.3 Software Challenges

Several requirements to simulation software which is applied in multiscale or scale transition scenarios have been mentioned in the previous sections; they are shown in Fig. 2.3 and are listed in the following:

- Single-scale solvers for each involved scale are required (yellow ellipse in Fig. 2.3). In the majority of “real-world” simulation scenarios, three-dimensional simulations are unavoidable. However, two-dimensional implementations can be very helpful during the code development and for various test scenarios as well as for realistic scenarios which allow for further model simplifications.
• Spatial adaptivity of particular solvers may be necessary to allow for simulations close to the scale transition (small-sized yellow-to-red-coloured ellipse in Fig. 2.3).

• The coupling of the single-scale solvers needs to be established (red ellipse in Fig. 2.3). Based on an accurate physical description, the respective implementation needs to comprise all required functionalities such as correct scaling of the parameters, mapping of degrees of freedom or boundary modelling. A respective implementation for hybrid Lattice Boltzmann–molecular dynamics simulations is discussed in Sec. 13.4. Alternatively, an extension of a single model may be applied which results in the applicability of the well-established original model beyond its default validity range (cf. Sec. 5.7).

Besides these particular challenges, additional challenges arise which are common in most software developments in the field of scientific computing. The simulation software shall comply with most common aspects from the field of software engineering (blue ellipse in Fig. 2.3). A modularised software design allows for a high level of flexibility, for example in terms of exchangeability of single software components. The software needs to be reusable in order to reduce future implementation efforts when building new simulation software on top of the existing components. Besides, the maintainability of the code must be guaranteed. A particular aspect in developing software for simulation research arises from the fact that new algorithms or corresponding improvements are published over and over. The incorporation of new algorithms into an existing piece of software is hence highly desirable which corresponds to the requirement of software extensibility.

The last category of relevant software challenges is given by high-performance computing (green ellipse in Fig. 2.3). As pointed out in Sec. 2.1, computational costs play a crucial role in flow simulations. This holds for single-scale considerations and also for various multiscale problems. Highly efficient single-scale codes are thus required as well as efficient multiscale simulation codes; following the discussion in Sec. 2.1, the two most efficient single-scale strategies do not necessarily yield the most efficient multiscale scheme. Hence, a particular investigation of the coupled software is required in this case. Besides, distributed or shared memory parallelisation of the codes and the respective coupling implementation are necessary to address large-scale problems. In hybrid simulation scenarios such as the ones described in Sec. 2.2, only a small region of the overall computational domain is solved by the fine-scale method. On the one hand, the physical accuracy with respect to fine-scale effects is limited by the size of this region. On the other hand, the size of the fine-scale region is limited by the available computational resources. As a consequence, the size of this region needs to be chosen as big as possible to increase physical accuracy and as small as necessary so that the computational costs are still affordable. Parallelising the hybrid simulation hence weakens the second limitation and thus allows for large-scale and physically more accurate simulations.

Concluding, various challenges arise with respect to the simulation software for concurrent multiscale simulations and simulations close to scale transition. Many of these challenges affect each other and often are conflicting; hence, a perfect solution which fulfills all challenges is hardly possible. In the following, different software packages for single-scale flow simulations or coupling flow simulations on different scales are reviewed and shortly discussed with respect to their functionality and the challenges from this section.

2.4 State-of-the-Art Software for Multiscale Fluid Dynamics Applications

The discussion on available software for multiscale flow simulation is restricted to pieces of software that deal with at least one of the considered scales. The listing of software packages is not complete due to the huge number of codes that are used within the CFD community. In case of single-scale considerations, the focus is mostly put on well-known simulation frameworks and community codes. In contrast, when it comes to concurrent coupling of different scales, there is hardly any sophisticated coupling software available which is widely distributed and represents a common choice within the CFD community. This might be due
to the complexity in the coupling schemes or the particular needs of every single project on multiscale fluid dynamics. Hence, a listing of all different codes/groups and a respective analysis could fill a book on its own. For this reason, only the most relevant pieces of multiscale or coupling software are briefly introduced within this section.

Starting with single-scale solvers for molecular dynamics, one of the most common simulation frameworks is given by LAMMPS\(^4\) [145]. This framework has been developed at the Sandia National Laboratory and supports sequential and massively parallel MD simulations. The software comprises amongst others implementations of *stochastic rotational dynamics*, a particle-based approach to mesoscopic flow simulations. LAMMPS has already been used in atomistic–continuum couplings for structural dynamics problems, cf. the website of LAMMPS. The extensible simulation package ESPResSo \(^5\) [115] represents another alternative for sequential or massively parallel molecular dynamics simulations. Besides the molecular dynamics solver, a non-adaptive Lattice Boltzmann solver is contained in the package. Both MD and LB solvers can be coupled in the sense of particle-fluid interaction, that is molecules are considered as particles that are suspended in the Lattice Boltzmann flow field. To the author’s knowledge, no concurrent coupling of Lattice Boltzmann and molecular dynamics in the sense of refining the fluid description has been reported yet. Another framework for molecular dynamics simulations is given by MarDyn \(^6\) [20]. It allows for three-dimensional massively parallel simulations and has mostly been used for micro- and nanoflows as well as other applications from chemical engineering. Similar to LAMMPS, MarDyn scales very well on peta-flop machines. A vectorised version of MarDyn is under current development allowing for highly efficient MD simulations.

Since it is Lattice Boltzmann simulations that are considered in the following within the context of mesoscopic flow simulations, the discussion on software packages for mesoscopic flow simulations is restricted to this respective subgroup. VirtualFluids \(^7\) [59] is a Lattice Boltzmann framework which has been developed at the TU Braunschweig. It allows for massively parallel Lattice Boltzmann simulations on spatially adaptive grids. The software has been used amongst others for free surface flows or fluid-structure interaction. No extensions of VirtualFluids to further approach the molecular regime or couplings with coarse-grained solvers have to the author’s knowledge been developed so far. The widely applicable Lattice Boltzmann (solver) from Erlangen named waLBerla \(^8\) does not support adaptivity yet. However, the regular Lattice Boltzmann solver allows for massively parallel simulations on peta-flop machines using a patch-based domain decomposition approach and shows very good scaling on up to 300k cores. Various complex flow simulations are incorporated such as particle-laden flows, free surface or blood flows including several for micro- and nanofluids. Furthermore, hybrid CPU-GPU simulations are supported \(^9\). LB3D\(^\circ\) is a Lattice Boltzmann code with major focus on multiphase and multicomponent flows as well as particulate flows. Similar to waLBerla, it shows good scaling behaviour on up to 300k cores. A fluctuating Lattice Boltzmann model has recently been incorporated. The code does not support spatial adaptivity. Palabos\(^5\) and OpenLB\(^5\) are two well-known community codes for three-dimensional Lattice Boltzmann simulations. Palabos was branched from the original OpenLB project in 2009. It covers amongst others thermal flows (based on the Boussinesq approximation), single-component multiphase, multicomponent multiphase and free surface flows. It operates on non-adaptive grids or—since version 0.7—on spatially adaptive grids. The latter, however, is supported for two-dimensional simulations only. The OpenLB project comes with similar features. To the author’s knowledge, both codes do not contain specific functionality for coupled flow simulations on multiple scales or model extensions for rarefied gas flows yet. Besides, the commercial Lattice Boltzmann solver PowerFLOW by Exa Cooperations\(^6\) shall be mentioned here. Its main application so far is found amongst others in automotive industries as Navier-Stokes alternative. Due to its high license costs, it is not subject of further considerations. From this review on existing

\(^4\)http://lammps.sandia.gov
\(^5\)http://crgforge.cse.cf.ac.uk/gf/project/lb3d/
\(^6\)http://www.palabos.org
\(^7\)http://www.openlb.org
\(^8\)http://www.exa.com
Lattice Boltzmann implementations, it can be concluded that all pieces of software come with particular features. However, there is no code which already comes with all features required, that is (dynamic) spatial adaptivity or micro- and nanoflow model extensions. In order to consider flows on the continuum/macroscopic scale, Navier-Stokes solvers represent the most common choice. A multitude of respective simulation packages exists; one of the most widely used simulation toolboxes is the open-source project OpenFOAM\textsuperscript{9}. It comes with a long list of solvers (e.g. compressible, incompressible or multiphase solvers) and supports unstructured meshes; for the \texttt{snappyHexMesh}, the underlying data management performs automatic load-balancing of the grid in case of massively parallel simulations and thus yields an approx. uniform distribution of the cells across the processors. OpenFOAM further comprises a molecular dynamics solver. However, no specific documentation of this solver is available in the current documentation available on the website of OpenFOAM as of Dec 10 2012. Besides, numerous commercial Navier-Stokes solvers are available such as CFX by ANSYS\textsuperscript{10}. Due to the license costs, they should not be part of more detailed considerations at this point despite the functionality they come with.

With spatial adaptivity playing a crucial role in many multiscale fluid dynamics scenarios, different software frameworks shall also be discussed which support (dynamic) spatial adaptivity. Deal II\textsuperscript{11} is a finite element-based framework using locally refined grids. The grid structure consists of quadrilateral (2D) or hexahedral cells (3D). Deal II allows for massively parallel simulations on these meshes on thousands of cores. The framework Peano [178] is based on similar grid structures. The traversal of the hexahedral cells is accomplished following the iterates of the space-filling Peano curve. This yields a high data locality and cache efficiency. Besides, dynamic spatial adaptivity is “for free” due to the strictly local space-filling curve-based grid traversal. Peano comes with different partial differential equation solvers such as solvers for the heat equation or the Navier-Stokes equations [133]. The latter comprises different time stepping and spatial discretisation schemes in two and three dimensions and also operates on dynamically changing spatially adaptive grids. DUNE\textsuperscript{12}, the distributed and unified numerics environment, is a framework for solving partial differential equations on various kinds of grids. DUNE is divided into several modules with its grid-interface “dune-grid” representing one of DUNE’s particular features: several implementations of DUNE’s grid interface are available and allow for the application of a discretisation scheme—such as finite elements or finite volumes—on different grid structures; the respective implementation of the scheme is left to the application developer or may be chosen from existing implementations of the PDELab module. Other discretisations such as finite difference methods are possible as well.

In contrast to the great variety of single-scale solvers for micro-, meso- and macroscopic scales, only a very limited number of multiscale coupling codes are available. For example, only few publications address Navier-Stokes–Lattice Boltzmann coupling schemes, cf. Chap. 12, and no general piece of software to couple these approaches has thus been developed yet. Several aspects on multiscale methods are picked up in [54]. Chapter 12 of this book focuses on a general strategy to define multiscale software using existing single-scale solvers and supporting adaptive simulation control methods. Many important aspects and general coupling approaches are re-visited. However, no particular information is provided in this chapter on parallel executions of the coupling software. Although a coupling to the molecular dynamics package LAMMPS is established with this piece of software, only little information on the coupling scheme and its complexity is presented. An evaluation of the capabilities of the software is therefore difficult. A general coupling framework has been presented in [35] for molecular–continuum flows. It allows to exchange data fields between the two separate simulations. The particular coupling routines that are required by the single solvers to incorporate the exchanged physical quantities, however, are included in each specific solver. Every solver thus needs to provide its own implementation of these mechanisms which may yield significant code duplications, cf. also the discussions in Chap. 13.

\textsuperscript{9}http://www.openfoam.org

\textsuperscript{10}http://www.ansys.com/Products/Simulation+Technology/Fluid+Dynamics/ANSYS+CFX

\textsuperscript{11}http://www.dealii.org

\textsuperscript{12}http://www.dune-project.org
to incorporate functionality into molecular dynamics simulations using LAMMPS is shown in [58]. Examples are shown for single-core and parallel scenarios where LAMMPS is either used as a library or a stand-alone application. However, only simple operations are invoked on the molecules such as modification of the force field or adding a constant displacement to all molecules. No steps towards concurrent molecular–continuum simulations have been taken in this publication.

Based on the available software from above, several conclusions can be drawn. With respect to molecular dynamics software, many sophisticated packages are available. Due to the active developments and past experiences with MarDyn, this simulation framework has been chosen for prototype implementations of a coupled Lattice Boltzmann–molecular dynamics scheme, cf. Sec. 13.3.

Although numerous Lattice Boltzmann implementations with various features already exist, none of them already comprises all the requirements that have been mentioned before such as spatial adaptivity or model extensions towards the molecular scale. Several possibilities exist in this case, amongst others:

1. Create a new LB solver which comes with all the respective requirements.
2. Extend a spatially adaptive LB code by the additional model extensions.
3. Extend a non-adaptive LB solver which already includes some model extensions by spatial adaptivity.
4. Reuse existing data structures to establish the respective specialised LB solver.

The first approach requires high implementational efforts and to the author’s opinion yields “just another solver with features A and B”. In contrast, providing extensions to an existing implementation yields less implementational work as long as the design of the underlying implementation allows for these extensions. For example, if a Lattice Boltzmann code is designed to work for regular, that is non-adaptive, grids only, then an extension to adaptive grids can become very difficult or even impossible without breaking the original code design. Hence, as a compromise between the first, second and third approach, one can reuse existing data structures and introduce the specialised solver based on these data structures. The data structures may stem from an existing framework implementation. On the one hand, they should be flexible enough so that the specific functionality of the respective (LB) solver can be incorporated. On the other hand, they should simplify the process of the solver development by providing implementations for some of the solver requirements. Since the Peano framework comes with spatially adaptive Cartesian grids and supports dynamic adaptivity as well as support for shared and distributed memory parallelisation, the decision was made to establish a Lattice Boltzmann solver within this piece of software.

Another advantage of the Peano framework is that it already contains a Navier-Stokes implementation. This solver can hence be reused for continuum flow problems or for hybrid Lattice Boltzmann–Navier-Stokes simulations.

Re-considering the scale separation map from Fig. 2.1, an open question still persists in terms of the setup of hybrid Lattice Boltzmann–molecular dynamics simulations. As pointed out in the discussion on multiscale and coupling software, only one general coupling framework [35] has been presented so far for this purpose. However, this framework requires that the specific coupling functionality is provided by the two solvers; its major focus lies on the pure data exchange of the flow quantities. In order to further facilitate the development and efficient execution of hybrid Lattice Boltzmann–molecular dynamics simulations, a coupling software for these hybrid simulations should

- comprise implementations of the most important coupling functionalities; cf. Sec. 13.2 for a detailed description of respective coupling components,
- allow for a simple exchange of existing coupling functionalities,
- support massively parallel hybrid simulations.

Since—to the author’s knowledge—no respective coupling software is available yet, the implementation of a new coupling software is required.
3 Aims and Outline

Having reviewed available pieces of software for flow simulations from micro- to macroscales, the aims and a brief outline of this thesis are given in the following. Two major aims can be formulated.

First, a Lattice Boltzmann simulation is incorporated into the Peano framework. This extends the range of applicability of the framework to mesoscopic flow simulations in two and three dimensions on spatially adaptive grids. The capabilities and compatibility of Peano with respect to Lattice Boltzmann methods are discussed in detail. Besides, new algorithmic approaches to Lattice Boltzmann implementations are presented such as a memory-efficient streaming step implementation and a methodology to apply the Lattice Boltzmann method on dynamically changing grids. In order to approach micro- and nanoflows, two particular model extensions are incorporated into the software: a fluctuating Lattice Boltzmann model which captures Brownian motion effects [42] and an extension of the Lattice Boltzmann method to finite Knudsen numbers which allows for the simulation of rarefied gas flows [114, 177]. Since Peano already comprises a Navier-Stokes implementation, a new technique to spatially couple Lattice Boltzmann and Navier-Stokes solvers is presented. Based on the new coupling approach, hybrid flow simulations become possible within the framework and extend its range of applicability from the pure continuum down to the micro- and nanoflow regime. In this context, technical aspects of coupling different solvers within Peano are discussed.

Second, hybrid Lattice Boltzmann–molecular dynamics simulations are addressed to bridge the gap between the meso- and the microscopic scale. As explained in the previous chapter, no coupling software for hybrid Lattice Boltzmann–molecular dynamics simulations which comprises a common implementation for all required coupling functionalities is available yet. For this reason, the macro-micro-coupling tool has been developed within the scope of this thesis. It is written in C++ and allows for massively parallel Lattice Boltzmann–molecular dynamics simulations in two and three dimensions. The different components of these hybrid schemes are separated from each other, and a selection of respective component implementations is provided in separate modules. Due to this modularised approach, the extension of the coupling tool by new component implementations is greatly facilitated. Using the coupling tool, the coupling of arbitrary Lattice Boltzmann and molecular dynamics simulations is accomplished via implementing four simple interfaces and incorporating calls to the coupling mechanisms into the two solver algorithms. The development of the macro-micro-coupling tool is extensively described, and a coupling of molecular dynamics and the Peano-based Lattice Boltzmann simulation is presented. Thus, flow simulations on micro-, meso- and macroscopic scales as well as hybrid simulations in the scale transition regime become possible with Peano.

The thesis is organised in five parts: the introduction (Part I, chapters 1–3), a review on the mathematical and physical theory of micro-, meso- and macroscopic flow descriptions (Part II, chapters 4–7), a detailed description of developed algorithms and respective implementations (Part III, chapters 8–13), the presentation of numerical results for single- and multiscale applications from computational fluid dynamics (Part IV, chapters 14–17) and a short summary (Part V).

In Part II, the mathematical and physical theory on micro-, meso- and macroscopic flow modelling is reviewed. The review focuses on molecular dynamics, Lattice Boltzmann methods and the incompressible Navier-Stokes model since these models are used throughout the following chapters. Each model is explained including its numerical realisation, and its features and limitations are shortly discussed. Molecular dynamics is described in Chap. 4. With the Lattice Boltzmann method representing a major topic of this thesis, particular emphasis is put on the description of this method in Chap. 5. The Navier-Stokes equations and their numerical treatment are presented in Chap. 6. The part closes with connecting the different single-scale descriptions to each other; the relations between the models are derived in Chap. 7.

Single- and multiscale algorithms and corresponding implementations are subject of Part III. Two molecular dynamics implementations—the framework MarDyn and a molecular
dynamics solver which is built into the macro-micro-coupling tool for testing and validation purposes—are described in Chap. 8. In chapter 9, a short introduction to the Peano framework is given. The Lattice Boltzmann implementation within Peano is presented in Chap. 10. Besides the discussion of efficiency and compatibility aspects, the overall design of the simulation and its sub-components is explained. A particular focus is put onto the implementation of the spatially adaptive Lattice Boltzmann scheme including a dynamic mesh refinement technique. The discussion of the single-scale implementations is closed with a short review on Navier-Stokes simulations in Peano, cf. Chap. 11. A new approach for spatially coupling Navier-Stokes and Lattice Boltzmann simulations is derived in Chap. 12. Based on this approach, different hybrid Lattice Boltzmann–Navier-Stokes schemes are developed, amongst others a hybrid method for hierarchical particle transport modelling. In addition, technical details on simulation coupling within Peano are explained. In the last chapter of Part III, that is Chap. 13, the macro-micro-coupling tool for hybrid Lattice Boltzmann-molecular dynamics simulations is described. The mechanisms to exchange flow quantities between Lattice Boltzmann and molecular dynamics are reviewed, and the arising software requirements for the coupling tool are pointed out using a hybrid prototype implementation. Based on this analysis, the coupling tool is developed, and its software design is discussed.

With the implementations for flow simulations on different scales available, numerical results for various applications from the field of fluid dynamics are presented in Part IV. The different applications are chosen according to the scale separation map from Fig. 2.1. The Peano-based Lattice Boltzmann simulation is validated in Chap. 14 using various benchmark scenarios. Besides, the sequential and parallel performance of the simulation are discussed. Results from simulations in the finite Knudsen range, that is for rarefied gas flows, are given in Chap. 15. The required model extension for the Lattice Boltzmann method is validated and subsequently applied to flows in microreactor geometries. The simulation of particle transport in so-called drift ratchets is subject of discussion in Chap. 16. The dynamic mesh refinement technique that has been developed in Chap. 10 is used to track and accurately resolve the particle geometry inside a complex pore geometry. In order to allow for the simulation of longer time scales in these particle simulations, the hierarchical particle transport model which combines Navier-Stokes and Lattice Boltzmann simulations (cf. Chap. 12) is applied. The part closes with numerical results for hybrid Lattice Boltzmann–molecular dynamics simulations, cf. Chap. 17. The functionality of the macro-micro-coupling tool is validated, and the sequential and parallel performance of the coupling tool is evaluated. A major issue of LB–MD schemes consists in the modelling of the open boundaries in the molecular dynamics simulation. A particular model based on radial distribution functions [180] (RDFs) is investigated in detail within the same chapter. Besides, Lattice Boltzmann–molecular dynamics simulations in channel-like simulation scenarios are presented. Finally, the major outcomes of the thesis are summarised in Part V.
Part II

Theoretical Foundation: Modelling Flows on Different Scales

In order to span the ranges of the scale separation map from Fig. 2.1, three different modelling approaches to fluid dynamics are reviewed: molecular dynamics [148], the Lattice Boltzmann method [163] and the Navier-Stokes description [70]. In Chap. 4, molecular dynamics is described. The basic algorithmic ideas are shortly addressed, including the linked cell scheme to speed up molecular dynamics computations. We basically use Eulerian methods to model flows on meso- and macroscopic scales which are described in Chap. 5 and 6. Special focus is put onto the Lattice Boltzmann method as it builds the core of the numerical results presented in Part IV. Techniques for particle-laden flows, the incorporation of fluctuating hydrodynamics and extensions to the range of finite Knudsen numbers will be presented within Chap. 5, pointing out the capabilities of the LBM on the meso- and close to the microscale. The macroscopic description of incompressible flows is given by the Navier-Stokes equations [162] and is outlined in Chap. 6. The link between these different levels of detail—micro-, meso- and macroscopic flow descriptions—is sketched in Chap. 7.

4 Microscopic Modelling: Molecular Dynamics

4.1 Model Description

The MD approach allows for modelling a fluid on the molecular scale. Therefore, single fluid molecules are described by means of their physical properties (e.g. their mass), their pairwise interactions and their respective motion over time according to Newton’s laws of motion [140]. Let \( x_p(t) \) denote the position of the molecule \( p \) at time \( t \), \( v_p(t) \) its velocity and \( m_p \) its mass. Then, the motion of a point-like particle over time can be described via ordinary differential equations for \( x_p \) and \( v_p \),

\[
\begin{align*}
\frac{dx_p}{dt} &= v_p \\
\frac{dv_p}{dt} &= \frac{1}{m_p} F_p,
\end{align*}
\]

(4.1)

and respective initial conditions \( x_p(t = 0) = x^0_p, \ v_p(t = 0) = v^0_p \). Besides, as it is only a fixed volume that shall be considered in a molecular simulation\(^2\), boundary conditions need to be applied to particles leaving the MD volume. Periodic boundary conditions are very popular (see Fig. 4.1) since they automatically yield the conservation of mass, momentum and energy and allow for the simulation of infinitely expanding domains. However, in case

---

1For more complex molecular models, such as molecules that consist of several atoms and thus lose their rotational symmetry, an additional set of equations is required to model the rotation and the torque. See Sec. 5.6 for details.

2We consider molecular systems in the NVT ensemble [71], that is systems where the number of particles, the considered volume and the temperature are kept constant.
Figure 4.1: (a) Periodic boundaries and linked cell discretisation. The computational domain is split into cells of size $dx_{lc} \geq r_c$. Particles leaving the domain on one side should enter the domain on the other side again. For this purpose, copies of the molecules are created in the dashed cells on the respective opposite side of the domain. (b) Sketch of the Lennard-Jones potential $U$ in dependence from $\|x_{p1} - x_{p2}\|$. In this case, the parameters $\epsilon$ and $\sigma$ are set to unity.

of hybrid LB–MD simulations, different boundary conditions are required that allow for the mapping of the conserved quantities between the LB and the MD solver region. Details on respective boundary treatments are described in Sec. 13.2.

The term $F_p$ represents all forces acting on the molecule $p$. Depending on the scenario to be modelled, different forces may become necessary to be included. For the purposes of the thesis at hand, the total force acting on a molecule can be split as follows:

$$ F_p = F^\text{interact}_p + F^\text{external}_p $$

where $F^\text{external}_p$ stands for external forces such as gravity and $F^\text{interact}_p$ represents forces arising from interactions between molecules. In this thesis, the popular Lennard-Jones (LJ) potential [130, 147] is used to model these intermolecular interactions. It is given by

$$ U^{LJ}(\|x_p - x_q\|) := 4\epsilon \left( \left( \frac{\sigma}{|x_p - x_q|} \right)^{12} - \left( \frac{\sigma}{|x_p - x_q|} \right)^{6} \right), $$

resulting in additive pairwise interaction forces

$$ F^\text{interact}_p := -\nabla_x \sum_{q \neq p} U^{LJ}(\|x_p - x_q\|) $$

between a particle $p$ and all other particles $q$. The first term of the potential in Eq. (4.3) models strong repulsive forces which hinder the nuclei to overlap (“Pauli repulsion”). The second term describes attractive forces for molecules at bigger distances (“van der Waals forces”). The parameter $\epsilon$ determines the depth of the potential whereas $\sigma$ is a characteristic length that determines the zero-crossing of the potential [71]. Both parameters strongly depend on the properties of the underlying fluid; see Tab. 4.1 for parametrisations of different fluids and Fig. 4.1 for a sketch of the Lennard-Jones potential.

### 4.2 Discretisation and Numerical Schemes

Common discretisation schemes for the equation system (4.1) are given by the Störmer-Verlet methods [166]. In the codes used and developed within this thesis, the leapfrog variant of the Störmer-Verlet scheme

$$ v_p(t + \frac{1}{2}dt) = v_p(t - \frac{1}{2}dt) + \frac{dt}{m_p} F_p(t) $$
$$ x_p(t + dt) = x_p(t) + dt v_p(t + \frac{1}{2}dt) $$

(4.5)
<table>
<thead>
<tr>
<th>Substance</th>
<th>Molecular weight</th>
<th>$\sigma$ (Å)</th>
<th>$\epsilon/k_B$ (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H$_2$</td>
<td>2.016</td>
<td>2.915</td>
<td>38.0</td>
</tr>
<tr>
<td>He</td>
<td>4.003</td>
<td>2.576</td>
<td>10.2</td>
</tr>
<tr>
<td>Ne</td>
<td>20.180</td>
<td>2.789</td>
<td>35.7</td>
</tr>
<tr>
<td>Ar</td>
<td>39.948</td>
<td>3.432</td>
<td>122.4</td>
</tr>
<tr>
<td>Air</td>
<td>28.964</td>
<td>3.617</td>
<td>97.0</td>
</tr>
<tr>
<td>N$_2$</td>
<td>28.013</td>
<td>3.667</td>
<td>99.8</td>
</tr>
<tr>
<td>O$_2$</td>
<td>31.999</td>
<td>3.433</td>
<td>113.0</td>
</tr>
<tr>
<td>CO</td>
<td>28.010</td>
<td>3.590</td>
<td>110.0</td>
</tr>
<tr>
<td>CO$_2$</td>
<td>44.010</td>
<td>3.996</td>
<td>190.0</td>
</tr>
<tr>
<td>CH$_4$</td>
<td>16.04</td>
<td>3.780</td>
<td>154.0</td>
</tr>
<tr>
<td>C$_2$H$_6$</td>
<td>30.07</td>
<td>4.388</td>
<td>232.0</td>
</tr>
</tbody>
</table>

Table 4.1: Lennard-Jones parameters $\epsilon$, $\sigma$ for different substances. The values are taken from [15, 83].

and the velocity-Störmer-Verlet method

$$x_p(t + dt) = x_p(t) + dt \left( v_p(t) + \frac{dt}{2m_p} F_p(t) \right)$$

$$v_p(t + dt) = v_p(t) + \frac{dt}{2m_p} \left( F_p(t) + F_p(t + dt) \right)$$

are applied; the latter has the advantage that both position and velocity of a molecule are given at the same points in time whereas the leapfrog method uses a staggered time stepping, evaluating $x_p(t)$ and $v_p(t + \frac{1}{2}dt)$. The Störmer-Verlet methods satisfy both time-reversibility and symplecticity which are important properties of dynamics in microscopic systems; for further details, see amongst others [71].

Considering Eq. (4.1) and assuming a system of $N$ molecules, the overall complexity of the force evaluation is $O(N^2)$. This becomes computationally very expensive for huge particle numbers. In order to reduce the computational load, the linked cell algorithm [6] is applied: a cut-off radius $r_c$ is chosen, the computational volume is subdivided into cells of mesh size $dx^{lc} \geq r_c$ and interactions between two molecules $p, q$ are only considered for $\|x_p - x_q\| \leq r_c$. For molecules whose distance is larger than $r_c$, the forces and potentials are set to zero. For the special case $r_c = 2^{1/6}\sigma$, that is for purely repulsive forces, both intermolecular force and potential exactly vanish at $r_c$. For other choices of $r_c$, the cut-off procedure yields a discontinuity in the Lennard-Jones potential. It can therefore have a strong influence on the simulation, for example in terms of energy conservation. In particular for USHER-based particle insertions (see Sec. 13.2), a continuous potential landscape is required. One possibility to resolve the discontinuity is to use a shifted potential $U^{LJ}_{shifted}(\|x_p - x_q\|) := U^{LJ}(\|x_p - x_q\|) - U^{LJ}(r_c)$ which by construction exactly vanishes at distance $r_c$. The linked cell approach is valid for simulations where the consideration of short-range interactions is sufficient\(^3\). It further reduces the complexity of the force evaluation to $O(N)$.

### 4.3 Features and Limitations

Molecular dynamics yields a purely microscopic description of the fluid, modelling each molecule separately, and therefore describes the motion of a fluid on the molecular scale. Macroscopic results for flow problems can be recovered from molecular dynamics simulations using averaging and sampling techniques. The single-centred Lennard-Jones model is a rather simple model. More enhanced molecule models exist, incorporating other types of molecular interactions or internal degrees of freedom. The latter may be constructed using multiple Lennard-Jones sites per molecule [71]. Despite its simplicity, single-centred Lennard-Jones simulations still can be used for a broad range of applications: liquid-solid

\(^3\)The attractive part of the Lennard-Jones potential decays very fast (at power six). Long-range interactions do not play a crucial role in the following considerations.
phase transitions, crystallisation processes or nanoflow problems such as those considered in the next parts. However, due to modelling the molecular scale, only very small volumes can be simulated using MD. So far, the biggest molecular system involved about $O(10^{12})$ molecules [46, 64]—in case of the trillion atom simulation in [64], this resembles a physical volume of only approx. $16\,\mu m^3$.

MD simulations are bound by their computational load; memory requirements typically play a negligible role. The most expensive part of the simulation is the force evaluation and the respective distance computations between the molecules, cf. [19]. An efficient shared or distributed memory parallelisation of MD codes that are based on short-range molecular interactions can be achieved by domain decomposition methods: each process represents one sub-domain, computing the trajectories of all molecules located herein. Molecules leaving the region and molecules that are required for the evaluation of the molecular interaction potential near the sub-domain boundary need to be communicated/synchronised over the neighbouring processes, yielding simple direct-neighbour communication.

5 Mesoscopic Modelling: The Lattice Boltzmann Method

5.1 Introduction

A more coarse-grained description of fluids is given by mesoscopic modelling approaches. Based on statistical mechanical principles and kinetic theory, averaged quantities and particle distribution functions are used to describe and evaluate the behaviour of a fluid. In contrast to continuum methods (see Chap. 6), mesoscopic methods capture the physics of different molecular ensembles. Letting these ensembles interact with each other, intermolecular interactions and collision processes can still be captured and incorporated on the mesoscopic level.

Several mesoscopic methods have been proposed in literature. Dissipative particle dynamics [86] is a Langrangian method similar to molecular dynamics where clusters of molecules are considered and transported, using Newton’s equations of motion. These clusters interact with each other via interaction potentials. Besides, dissipative and random forces are included to allow for friction effects and diffusion of the fluid.

The Direction Simulation Monte Carlo (DSMC) method [14] is also a particle-based approach, mostly used to simulate rarefied gas flows. Each particle may represent several molecules. The standard algorithm consists of three parts: first, the particle positions and velocities are updated, following a simple time integration scheme. Second, the collisions between the particles are modelled via stochastic collision rules. In order to find pairs of “neighbouring” particles for the collision, the particles are sorted into cells; if particles reside in a common cell, they may contribute to the collision process. In the third step, boundary conditions are applied to the particles. Due to the randomness in the collision process, the convergence rate of Direct Simulation Monte Carlo methods is comparably slow, scaling in $\sqrt{N}^{-1}$ where $N$ denotes the number of particles in the simulation [98].

Another approach is given by the Lattice Boltzmann method (LBM), a grid-based mesoscopic method, and is described in more detail in the following. In Sec. 5.2, its principles and its underlying algorithm is reviewed. Information on the collision modelling is provided in Sec. 5.3, followed by an extension of the standard LBM algorithm to adaptive grids in Sec. 5.4. Afterwards, in Sec. 5.5, boundary conditions for the method are shortly explained. Closing the methodological description of the Lattice Boltzmann method, an extension of the scheme to the rarefied gas scheme is pointed out in Sec. 5.7. For further details on the LBM, the reader is referred to [2, 27, 163, 182].

5.2 Model Description

The Lattice Boltzmann schemes evolve from a particular simplified discretisation of the Boltzmann equation, see Chap. 7 for details on the approximation process. Therefore, space is discretised by a quadratic (2D)/cubic (3D) lattice. Probability densities or particle distribution functions $f_i(x,t)$, $i = 1, ..., Q$, are introduced and defined in the centre of each
lattice cell. Each density represents the probability to encounter molecules moving with the lattice velocity \( c_i \) within a small region around \( x \) at time \( t \). The macroscopic quantities, that is fluid density \( \rho(x,t) \) and flow velocity \( u(x,t) \), can be evaluated in each grid cell, corresponding to the zero- and first-order moments of the distribution functions:

\[
\rho(x,t) = \sum_i f_i \\
\rho(x,t)u(x,t) = \sum_i f_i c_i. \tag{5.7}
\]

The set of lattice velocities \( c_i \) and a time step \( dt \) are chosen such that molecules moving with velocity \( c_i \) exactly traverse one lattice cell. The Lattice Boltzmann equation models the convective-diffusive behaviour of the fluid,

\[
f_i(x + c_i dt, t + dt) = f_i(x, t) + \Delta_i(f - f^{eq}), \tag{5.8}
\]

where diffusion and convection are separated into two subsequent algorithmic steps—the collide step and the propagation or streaming step:

\[
f_i^*(x, t) = f_i(x, t) + \Delta_i(f - f^{eq}) \quad \text{Collide step}
\]

\[
f_i(x + c_i dt, t + dt) = f_i^*(x, t) \quad \text{Streaming step.} \tag{5.9}
\]

Here, the distributions \( f_i^*(x, t) \) denote the respective post-collision states. The operator \( \Delta_i(f - f^{eq}) \) mimics intermolecular collisions on the statistical level. Therefore, we assume our thermodynamical system to only deviate slightly from its equilibrium state, given in terms of the probability densities \( f_i^{eq} \), respectively. A discussion of collision operator models is given in Sec. 5.3. The update rule from Eq. (5.7) is illustrated in Fig. 5.1, using the common \( D2Q9 \) velocity discretisation (two-dimensional space, nine lattice velocities). Different formulations for the discrete representation of the equilibrium distribution \( f_i^{eq} \) have been proposed, see amongst others [7] and [79]. In the following, the standard polynomial form is used which is derived from the Maxwell-Boltzmann distribution in the low Mach number (Ma) limit, either via a Taylor expansion [80] or a projection onto the tensor Hermite polynomial basis [73, 158]:

\[
f_i^{eq}(\rho, u) = w_i \rho \left( 1 + \frac{c_i u}{c_s^2} + \frac{(c_i u)^2}{2c_s^4} - \frac{u^2}{2c_s^2} \right) \tag{5.10}
\]

where \( c_s \) denotes the speed of sound\(^4\) on the lattice, and \( w_i \) are the lattice weights, depending on the underlying set of lattice velocities. In order to stay consistent in the macroscopic, i.e. continuum, limit and to provide a sufficient level of isotropy on the mesoscale, the weights

\(^4\)The requirement of low Mach numbers hence reads: \( \text{Ma} = \frac{\|u\|}{c_s} \ll 1 \).
need to fulfill the following isotropy constraints \[42, 163, 182\]:

\[
\begin{align*}
\sum_i w_i & = 1 \\
\sum_i w_i c_i & = 0, \quad \alpha \in \{1, \ldots, D\} \\
\sum_i w_i c_i c_{i\alpha} & = c_\alpha^2 \delta_{\alpha\beta}, \quad \alpha, \beta \in \{1, \ldots, D\} \\
\sum_i w_i c_i c_{i\alpha} c_{i\beta} & = 0, \quad \alpha, \beta, \gamma \in \{1, \ldots, D\} \\
\sum_i w_i c_i c_{i\alpha} c_{i\beta} c_{i\gamma} & = c_\alpha^2 (\delta_{\alpha\beta} \delta_{\gamma\delta} + \delta_{\alpha\gamma} \delta_{\beta\delta} + \delta_{\alpha\delta} \delta_{\beta\gamma}), \quad \alpha, \beta, \gamma, \delta \in \{1, \ldots, D\}.
\end{align*}
\] (5.11)

From these constraints and the form of the equilibrium distribution, it follows:

\[
\begin{align*}
\sum_i f_{i\alpha}^{eq} & = \sum_i f_i = \rho \\
\sum_i f_{i\alpha}^{eq} c_{i\alpha} & = \sum_i f_i c_{i\alpha} = \rho u_{i\alpha} \\
\sum_i f_{i\alpha}^{eq} c_{i\beta} c_{i\gamma} & = p \delta_{\alpha\beta} + \rho u_{i\alpha} u_{i\beta}
\end{align*}
\] (5.12)

with the pressure \(p\) given by the relation \(p := \rho c_s^2\).

5.3 The Collision Operator: Assumptions and Models

The collision process between molecules can become very complex; exact solutions for the collision operator from Eq. (5.8) are hardly known. Several constraints can be imposed on the operator though:

1. The collision process needs to conserve mass, momentum and energy (conservation property).

2. According to the second law of thermodynamics, the underlying molecular system needs to tend towards equilibrium when undergoing successive collision processes. This also corresponds to an H-theorem yielding a non-decreasing entropy.

All collision models that are presented in the following yield local distribution modifications, i.e. only distributions \(f_i\) within a particular Lattice Boltzmann cell \(x\) interact with each other during the collision. From the conservation property and Eq. (5.12), it becomes apparent that the collision operator may only modify the non-equilibrium parts \(f_i^{neq} := f_i - f_i^{eq}\) of the distributions. The second property is tedious to be fulfilled in the discrete settings of the Lattice Boltzmann methodology. Although the entropic models [7] typically are derived from a respective H-theorem, this is not the case for most other models. Amongst others, Yong and Luo have shown that some popular models do even not admit an H-theorem [185].

Still, pushing the system locally towards equilibrium can be included in the collision model in the form of a local relaxation procedure. Several relaxation models for \(\Delta (f - f^{eq})\) exist [13, 39, 69], with the BGK (or single-relaxation-time) collision model [13] representing the most common choice:

\[\Delta^{BGK} (f - f^{eq}) = -\frac{1}{\tau} (f - f^{eq}).\] (5.13)

The relaxation time \(\tau\) is directly related to the kinematic viscosity \(\nu\) of the fluid,

\[\nu = c_s^2 dt (\tau - 0.5).\] (5.14)

Due to stability constraints and the requirement of positive viscosity, \(\tau\) is restricted, \(\tau > 0.5\). In the following, the relaxation parameter is always chosen such that

\[\tau \in (0.5, 2).\] (5.15)
The BGK model is very simple in its form and, respectively, efficient in terms of computational complexity. Due to its simplicity, several well-known deficiencies come with this model, such as the stability criterion from above, a strict coupling of bulk and shear viscosities or a fixed Prandtl number in case of thermal simulations [39, 186].

Another collision model which overcomes some of the aforementioned drawbacks of the BGK scheme is the multiple-relaxation-time (MRT) scheme [39]. Instead of relaxing directly each distribution function \( f_i \), the relaxation process is carried out in the moment space \( \mathbb{R}^Q \). Therefore, the non-equilibrium parts \( f_{\text{neq}} \) are mapped from the phase to the moment space via a linear mapping \( M : \mathbb{R}^Q \rightarrow \mathbb{R}^Q \) and each moment is relaxed separately, applying a diagonal relaxation matrix \( \Gamma \in \mathbb{R}^{Q \times Q} \). The relaxed moment contributions are mapped back to the phase space and the distribution functions can be updated, accordingly. The collision operator reads

\[
\Delta^{\text{MRT}}(f - f^{eq}) = M^{-1} \Gamma M (f - f^{eq}).
\] (5.16)

Due to the existence of several relaxation times, a particular tuning of the relaxation parameters is possible, yielding amongst others better stability properties compared to the BGK method [39, 106]. The overall computational costs of this model come at a respective overhead of approx. 10-20% [38, 39, 106].

Both the BGK and the MRT model yield the Navier-Stokes equations in the macroscopic limit, cf. Chap. 7. Besides, modelling the collision term in the Lattice Boltzmann equation may also be used to introduce new physical effects into the mesoscopic fluid description. One example for such effects is the extension to fluctuating hydrodynamics. Brownian fluctuations play a crucial role in various nanoflow scenarios such as colloidal suspension flows [103] or polymer translocations [60]. On the nanoscale, the impact of the molecular random motion becomes apparent and cannot be neglected anymore in the flow models. The numerical treatment of the underlying stochastic processes, however, is far from trivial. Several approaches exist [1, 41, 103, 159] and new methods are still part of current research.

A particular form of the MRT approach has been presented in [42], incorporating Brownian fluctuations into the collide step. During the relaxation process in the moment space, Gaussian noise is added to the non-conserved moments, yielding:

\[
\Delta^{\text{FLB}}(f - f^{eq}) = \Theta^{-1} M^{-1} \Gamma M (f - f^{eq}) + \Phi r \] (5.17)

with \( r \in \mathbb{R}^Q \) containing zero-mean unit-variance Gaussian pseudo-random numbers. The diagonal matrix \( \Phi \) holds the variances for the single fluctuating contributions. Dünnweg et al. describe in [42] that—in order to fulfill detailed balance\(^5\)—the variances need to fulfill \( \Phi_{ii}^2 + \Gamma_{ii}^2 = 1 \) for all moments. In a similar fashion, other additive force terms can be incorporated into MRT-based schemes, see for example [114]. Besides the transformation matrix \( M \), a diagonal matrix \( \Theta \in \mathbb{R}^{Q \times Q} \) incorporates a scaling of the distributions according to the temperature \( T \), Boltzmann’s constant \( k_B \), the mesh size \( dx \) and the time step \( dt \),

\[
\Theta_{ii} \propto \sqrt{\frac{k_B T dt^2}{c_s^2 dx^3}}.
\] (5.18)

A detailed analysis with respect to the macroscopic representation of the arising fluctuating terms has been carried out by Dünnweg et al., showing that their model yields the Landau-Lifshitz Navier-Stokes equations [107] in the continuum limit. Hence, their model is consistent with the continuum representation of thermal fluctuations on the one hand and fulfills detailed balance on the other hand which is essential on the microscopic scale.

---

\(^5\)Besides the zero- and first-order moments from Eq. (5.7), the moment space contains links to the second-order moments: the bulk and shear stresses. In addition, further degrees of freedom are available, corresponding to higher-order moments of no physical relevance. These moments are called \textit{ghost moments} or \textit{ghost modes}.

\(^6\)Detailed balance implies that the probabilities for an elementary process and its reverse process are identical. Amongst others, Boltzmann’s H-theorem uses this relation to show that entropy can only increase in isolated systems, cf. [182].
5.4 Adaptive Formulations

As mentioned before, the Lattice Boltzmann scheme typically uses a regular spatial discretisation based on Cartesian grids with quadratic or cubic grid cells. For big domains, a homogeneously refined grid, however, can become expensive with respect to both, computational costs and required memory. The latter plays a crucial role in Lattice Boltzmann simulations, since we need to store at least one set of particle distribution functions per grid cell which results in 9–27 doubles per cell (2D/ 3D). Thus, a spatially adaptive approach to LB simulations is highly desirable, using coarser resolutions in those parts of the computational domain where less accuracy is required. Before revising different adaptive formulations, a short review on the dimensional analysis in Lattice Boltzmann simulations is presented in the following, as this is of fundamental importance to the respective scalings in the adaptive case.

Similar to most simulation methods, most LBM implementations use a particular dimensionless form of the Lattice Boltzmann equation. Characteristic scales \( x^c \) are defined and all quantities \( x \) are scaled to dimensionless lattice values \( x_L \) via \( x := x L \cdot x^c \Leftrightarrow x_L = x/x^c \). The values \( x_L \) are used in the simulation. We therefore require characteristic values for length, time and density scales. These values are chosen—consistent to most LB formulations in literature—as the mesh size \( dx \), the time step \( dt \) and the average density \( \rho \) of the considered fluid; this directly implies that \( dx_L = 1 \), \( dt_L = 1 \) and \( \rho_L = 1 \). For the velocity and the kinematic viscosity, the dimensional analysis implies:

\[
\begin{align*}
  \vec{u}_L &= \vec{u} \cdot \frac{dx}{dx} \\
  \nu_L &= \nu \cdot \frac{dx}{dx^2}.
\end{align*}
\]  

(5.19)

It can further be shown that the dimensionless speed of sound is given by \( c_{xL} := \frac{1}{\sqrt{\nu}} \).

Stepping towards spatially adaptive schemes, the relations from Eq. (5.19) show the typical dilemma of adaptive Lattice Boltzmann schemes: introducing mesh sizes \( dx^l \) for different grid levels \( l \) and a constant refinement factor \( r := \frac{dx^{l+1}}{dx^l} \) yields that either the kinematic viscosity or the dimensionless velocity can be kept constant over all grid levels.

On the one hand, retaining the kinematic viscosity on all grid levels, that is \( \nu_L^l = \text{const.} \forall l \) (viscosity-scaling), results in a quadratic decrease in the time step size, \( dt_L = r^{-2} dt^{l+1} \).

Thus, a high number of grid levels implies extremely high computational costs. Besides, the speed of sound varies between the different grid levels, yielding a decrease of the Mach number on finer grid levels; see amongst others [149] for an adaptive LB scheme using the viscosity-scaling.

On the other hand, the velocity can be kept constant, meaning that \( \frac{dx^{l+1}}{dx^l} = \frac{dt^{l+1}}{dt^l} = r \) (velocity-scaling). The computational costs on finer levels consequently increase linearly with the refinement factor \( r \). Besides, due to the stability restrictions of the Lattice Boltzmann method, cf. Eq. (5.15), the number of grid levels that can be used in adaptive simulations, is restricted, see amongst others [53, 172] for respective grid level limits. Abstaining from both approaches, i.e. using the same time step on all grid levels with varying velocities and viscosities, implies an even more severe limitation of grid levels.

Most adaptive LB schemes make use of the velocity-scaling approach. Different methods have been proposed, providing various methods to convert the particle distribution functions between subsequent grid levels \( l-1 \) and \( l \). One can group the approaches into two categories: interpolation-based and volumetric schemes.

The interpolation-based methods assume a discretisation of space where each coarse grid lattice node on level \( l-1 \) which is located at the interface to the finer grid level coincides with a fine grid node on level \( l \) [53], cf. Fig. 5.2. After solving one time step on the coarse level, the blue-coloured particle distribution functions of the fine grid region are interpolated from the coarse grid values, using temporal and spatial interpolation. In order to obtain a second-order scheme, an interpolation of order \( \geq 2 \) is required. Filippova et al. use second-order time and space interpolation [53]; Yu et al. [187] apply a third-order spatial interpolation. Besides, schemes based on first-order interpolation and with slightly less accuracy have been developed [116]. Providing boundary conditions via this interpolation at the coarse-to-fine
grid boundaries in each fine grid time step, \( r \) time steps on the fine grid can be carried out. The resulting solutions in fine grid nodes which coincide with respective coarse grid nodes are transferred back to the coarse grid and another coarse grid time step can be performed. For both, the transfer of the distributions \( f_i \) from the coarse to the fine grid and vice versa, a re-scaling of the distributions is required to conserve the viscous stresses near the interface; it is typically directly incorporated into the interpolation/restriction step. The conservation of mass and momentum between the grid levels is fulfilled up to the respective interpolation accuracy.

In contrast to the interpolation-based methods which carry out the transfer directly on the boundary nodes of the respective grid levels, the volumetric schemes use an overlap region to map the distributions and allow for the conservation of mass, momentum and viscous stresses [25, 26, 150]. The respective algorithm is sketched in Fig. 5.3. The overlap region has a thickness of one coarse or \( r \) fine grid cells. The collision step is performed in all cells of the coarse grid, including the cells within this overlap. Interpreting the particle distributions as average density distributions of each volumetric grid cell, the post-collision particle densities \( f_i^{\text{post-coll}} \) are distributed in the fine grid cells of the overlap layer. The original algorithm uses a homogeneous distribution of the densities in the fine grid cells, i.e.

\[
f_i^{\text{final}}(x, t) = f_i^{\text{final}}(\bar{x}, t) \quad \forall x \in \text{Vol}(\bar{x})
\]

where \( x \) represents the centre of the respective fine grid cell, \( \bar{x} \) the centre of the coarse grid cell and \( \text{Vol}(\bar{x}) \) its volume. Similar to the interpolation-based methods, this yields a simulation scheme for fluid dynamic problems of first-order accuracy. An improvement to second-order can be achieved via interpolation of the distributions within the fine grid cells of the overlap region between neighbouring coarse grid cell volumes, see amongst others [25, 26]. Having defined the fine grid densities in the overlap based on the post-collision states of the coarse grid, the time stepping procedure takes place on the fine grid. The collide-stream algorithm is carried out in all fine grid cells outside the overlap whereas only streaming is applied in the overlap domain. The streaming operation transports the particle distributions from the fine grid overlap into the non-overlap region of the fine grid. Besides, particle distributions from the non-overlap region of the fine grid enter the overlap region. Performing this time stepping scheme \( r \) times on the fine grid yields a full set of particle distribution functions.
Collide on coarse level
Propagate to fine level

Collide on fine level
Stream on fine level

Restrict to coarse level
Stream on coarse level

Figure 5.3: Volumetric adaptive Lattice Boltzmann scheme [25, 137]. The overlap region consists of one coarse grid cell (or \( r = 3 \) fine grid cells, respectively) and is located in the centre of the one-dimensional illustration. First, the collide step is executed on all coarse grid cells and the coarse grid cells belonging to the overlap region. The post-collision distributions that point into the fine grid region (visualised by blue arrows) are distributed over all fine grid overlap cells. For all cells of the non-overlap fine grid, the collide-stream algorithm is performed three times whereas the fine grid cells within the overlap only participate in the streaming. The streaming transports the blue-coloured distributions from the overlap into the non-overlap fine grid domain and fills the fine grid cells in the overlap with the green-coloured distributions. After the three fine grid time steps, the green-coloured distributions are averaged and written to the coarse grid. At the same time, the streaming operation is applied to all coarse grid cells, transferring the yellow-coloured distribution into the non-overlap coarse grid region. Storing the averaged green-coloured distribution inside the coarse grid cell of the overlap yields a complete set of particle distributions on the coarse grid for the next time step.

in the overlap region which is oriented towards the coarse grid\(^7\). These distributions are averaged and transferred to the coarse grid. Executing the streaming step for the original post-collision distributions \( f_i^{\text{post}} \) and storing the averaged values from the fine grid in the respective coarse grid cell yields a complete set of distributions on the coarse grid, and the next coarse grid time step can be carried out. One particular feature of the volumetric schemes is the exact fulfillment of the conservation laws for mass and momentum. For the homogeneous distribution of the coarse grid populations to the fine cells, this property is trivial. For the second-order volumetric schemes which use interpolation techniques similar to the purely interpolation-based schemes, a more detailed analysis is necessary; still, exact conservation is guaranteed also in this case. For example, in [25], Chen et al. introduce a state-flux function \( F_i(x', x) \), describing the flux between neighbouring volumes \( x' \) and \( x \). After deriving the required interpolation rules, they modify the state-flux function for the “self-advection” \( F_i(x, x) \), so that the conservation laws are enforced. Besides, no temporal interpolation is needed, yielding lower memory requirements with respect to the storage of the particle distribution functions from previous time steps.

5.5 Boundary Conditions

Similar to other grid-based solvers for differential equations, boundary conditions need to be provided in Lattice Boltzmann schemes, accounting for the finite size of the computational domain. The general idea is sketched in Fig. 5.4: each particle distribution function that

---

\(^7\)The term “full set which is oriented towards the coarse grid” in this case means that every fine grid overlap cell, from which a distribution \( f_i \) is streamed into the non-overlap fine grid region over \( r \) fine grid time steps, obtains a distribution \( f_i^r \) from the non-overlap fine grid, with \( f_i^r \) pointing towards the coarse grid domain.
Figure 5.4: Boundary treatment in Lattice Boltzmann schemes. A boundary is located at the top of the domain. Particle distributions (depicted by black arrows) that enter the domain from the boundary during streaming need to be constructed by applying a suitable boundary condition.

would enter a lattice cell from outside the computational domain in the streaming step needs to be constructed via a suitable boundary treatment. The derivation, construction and analysis of boundary conditions in the context of Lattice Boltzmann simulations is far from trivial and challenging enough to fill a separate PhD thesis [184]. Much research has been and is still spent on this topic, so for example in the context of second-order boundary conditions or non-reflecting conditions for open flow systems [91].

In the following, boundary conditions for different domain boundary types—no-slip and moving walls, periodic boundaries, channel-inlet and -outlet boundaries—are reviewed. Most of these conditions provide a first-order accurate boundary treatment. They are very efficient from the computational point of view as the respective treatment of the distributions is a strictly local process, i.e. no neighbouring information is required. For this reason, they are a common choice in many Lattice Boltzmann codes.

No-slip conditions are used to model fixed walls or obstacles within the flow. The fluid is assumed to stick to these objects, resulting in a vanishing flow velocity on their surfaces. The respective boundaries are handled by the half-way bounce-back scheme [163, 182]. If a distribution $f_i$ would enter the computational domain from a solid non-moving wall during streaming, it is constructed by its post-collision counterpart $f_i^*(x, t)$ from the same cell:

$$f_i(x, t + dt) = f_i^*(x, t). \quad (5.21)$$

Mass is automatically conserved by this method. Besides, the half-way bounce-back scheme is second-order accurate for a wall which is located exactly half-way between neighbouring grid cells. For other cases, it deteriorates to a first-order method. Several extensions to second-order accurate boundary treatment have been proposed, see amongst others the introductory discussions in [176]. Most of these extensions, however, require information from the neighbourhood of the considered lattice cell. The half-way bounce-back scheme can be applied without the necessity of other information than the cell-local distributions. For this reason, it is also often used in the context of simulations in complex geometries such as porous media.

Moving wall boundaries are treated similarly to the no-slip case and model—as the name suggests—moving objects. Due to the velocity $u_w$ of the respective object, an additional acceleration term needs to be taken into account [103]:

$$f_i(x, t + dt) = f_i^*(x, t) + \frac{2}{c_s^2} w_i \rho_w (c_i \cdot u_w). \quad (5.22)$$

where $\rho_w = \rho(x, t)$ is chosen to be the density of the current grid cell.

Typical examples where moving boundaries play a crucial role include cavity, Couette or

\*Pressure waves are often encountered in Lattice Boltzmann simulation schemes due to its explicit and weakly compressible nature. Non-reflecting boundary conditions try to reduce this issue by letting the waves escape the computational domain.
particulate flows, that is flows containing one or several particles; see Sec. 5.6 for a closer discussion of the latter. In cavity scenarios, a box is filled with fluid and the upper lid is constantly moved from left to right, accelerating the fluid within the box and creating vortices within the box. Exemplary flow fields are shown in Fig. 5.5. In (two-dimensional) Couette flows, a channel-like setup is considered: Periodic conditions are applied at the in- and outlet of the channel, a no-slip wall is used at the bottom and a moving wall with velocity $u_w$ is modelled at the top boundary. As the fluid needs to stick to both the moving and the pure no-slip wall, its velocity decreases linearly from $u_w$ to 0 along the perpendicular flow direction.

In order to simulate open systems such as channels or jets, periodic, velocity or pressure conditions are required. The simulation of periodic domains is straightforward and can be established analogously to the periodicity in MD simulations, cf. Fig. 4.1: particle distributions that would leave the computational domain on one side during streaming are inserted into the boundary cells on the other side of the domain. Velocity conditions are used to prescribe a given velocity profile in a particular boundary region. In the following, these conditions are applied by setting the missing distributions $f_i$ to the respective equilibrium states:

$$f_i(x, t + dt) = f^{eq}_i(\rho, u_{in}) \quad (5.23)$$

where $u_{in}$ is the prescribed velocity inlet profile.

Setting the equilibrium state resembles a first-order method. Second-order accurate conditions have been developed and described in literature. Most of them are limited to geometrically simple velocity boundaries. One method for straight velocity boundaries is provided by Zou and He [190]. They use the prescribed velocity $u_{in}$ and the already existing distributions to compute the expected density $\rho$ within the grid cell. Subsequently, they determine the equilibrium distribution and construct the missing distributions $f_i$ via a bounce-back scheme which is applied to the non-equilibrium parts $f_i$. An extension to the method of Zou and He was presented in [82], allowing for arbitrary inflow directions.

A different approach to velocity boundary conditions is given by extrapolation schemes, see amongst others [28]. These, however, require valid quantities in neighbouring stripes of lattice cells within the inner region of the computational domain; see [111] for a review.

---

3. Analogous simulations can also be conducted in three-dimensional setups. Here, periodic conditions are also applied at the other two channel boundaries. The Couette flow thus resembles a flow between two infinitely sized plates of which one is moving.
and evaluation of different second-order velocity boundary methods and [184] for a closer analysis of different dirichlet-type boundary conditions. Besides the velocity inlet and periodic conditions, pressure conditions are frequently used and required in many engineering scenarios, so for example to account for pressure drops or to define valid outflow conditions in channel flow scenarios. As the pressure is proportional to the fluid density in Lattice Boltzmann simulations, it is sufficient to specify a particular value for the density at the pressure boundary. In order to locally prescribe a certain density $\rho_{\text{out}}$, the approach from [100] is to be used:

$$f_i(x, t + dt) = f_i^{eq}(\rho_{\text{out}}, u) + f_i^{eq}(\rho_{\text{out}}, u) - f_i^*(x, t).$$

(5.24)

For an overview and evaluation of further outflow conditions, see amongst others [94, 184].

5.6 Moving Obstacles

An extended boundary treatment is required when flow systems with suspended particles need to be considered. On the one hand, the particle movements are dictated by the flow field. The momentum of the fluid acts onto the particle, and the particle feels the respective exerted force and gets dragged along the flow field. On the other hand, the particle movements may induce changes to the flow field since the fluid near the particle’s surface is accelerated according to the velocity at this surface. Depending on the intensity of the particle’s influence onto the flow, a one- or two-way coupling is required to solve the underlying physical problem. In the following, a two-way coupling method [88, 103, 104] is described for isolated, rigid spherical particles; a particular solution to one-way coupling strategies is described in Sec. 6.3 in the context of macroscopic flow simulations.

First, a suitable mathematical model for the particles and their movements needs to be supplied. Analogous to the mathematical description of the molecules in Sec. 4.1, the movement of a rigid particle is characterised by Newton’s equations of motion:

$$\frac{dx}{dt} = v_p$$

$$\frac{dv}{dt} = \frac{1}{m_p} F_p.$$  \hspace{1cm} (4.1)

In contrast to the molecular description, however, the particle under consideration is not a point particle anymore and fills a certain volume in space. Rotational forces may occur and need to be incorporated into the model by two additional ordinary differential equations, relating the orientation $R_p(t)$, the angular velocity $\omega_p(t)$, the angular momentum $L_p(t)$ and the torque $\tau_p(t)$ [9, 131]:

$$\frac{dR_p}{dt} = (\omega_p \ast) R_p$$

$$\frac{dL_p}{dt} = \tau_p$$

(5.25)

where the asterisk operator $\ast : \mathbb{R}^3 \rightarrow \mathbb{R}^{3 \times 3}$ transforms a three-dimensional vector $a$ into matrix form as follows:

$$a \ast := \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} \ast = \begin{pmatrix} 0 & -a_2 & a_1 \\ a_2 & 0 & -a_0 \\ -a_1 & a_0 & 0 \end{pmatrix}.$$  \hspace{1cm} (5.26)

The total torque is given by the sum over all torque contributions that act onto different positions $x_i(t)$ of the particle surface; each torque contribution is given by the cross product of the distance $(x_i(t) - x_p(t))$ of the surface point from the centre of mass $x_p(t)$ and the respective force $F_i(t)$:

$$\tau_p(t) = \sum_i (x_i(t) - x_p(t)) \times F_i(t).$$  \hspace{1cm} (5.27)

10 A method to solve fluid-particle simulations provides a one-way coupling of the flow and the particle if the flow field acts onto the particle, but the particle does not influence the flow. In contrast, two-way couplings allow the flow to influence the particle movement and vice versa.

11 The methodology can be completely transferred to particles of more complex shape.
The relation of the angular velocity and the angular momentum is given by

$$\omega_p(t) = (R_p(t)I_pR_p^T(t))\mathbf{L}_p(t)$$

(5.28)

where $I_p$ is the body space\(^1\) inertia tensor \([9]\) which is initially given in form of a diagonal matrix for spherical objects, see for example \([131]\). With the Eqs. (5.27),(5.28), the differential equation system from above, i.e. Eqs. (4.1),(5.25) is closed and may be solved by Verlet-like or Euler time integration schemes, see Sec. 4.2 and \([45]\).

In order to relate both forces exerted by the fluid onto the particle and vice versa, the Eulerian description of the hydrodynamic Lattice Boltzmann solver and the Lagrangian specification of the particle motion need to be set into relation. The respective methodology is illustrated in Fig. 5.6: the particle is assumed to span over several LB grid cells. The grid cells whose centre is located within the particle geometry are considered to lie outside the flow domain and are referred to as outer cells whereas cells that are filled with fluid are called inner cells. By this procedure, the particle geometry is mapped onto the Lattice Boltzmann grid.

The force contributions $F_i(t)$ from Eq. (5.27) which arise from the hydrodynamic interaction of the fluid with the suspended particle are extracted from the Lattice Boltzmann data. Different methods have been proposed for this procedure. Similar to Navier-Stokes-based schemes where only discrete pressure and velocity values and their derivatives can be extracted from the simulation data, the stress integration approach approximates the force acting onto the particle via integration of the fluid stress tensor $pI + \rho\nu(\nabla_u u + (\nabla_u u)^T)$ on the particle surface. The tensor may be evaluated applying an extrapolation of the gradients onto the surface or via a local evaluation of the total tensor based on the non-equilibrium parts of the distributions, cf. \([127, 186]\). Another alternative to determine the local force contributions is given by the momentum exchange method \([103, 104]\), see Fig. 5.6: during

\(^1\)In rigid body dynamics, the world space denotes the global space including a respective coordinate system in which all considered objects are located whereas the body space corresponds to the local coordinate system of each body. Therefore, the body space is always aligned with the current orientation $R_p(t)$ of the object and $R_p(t)$ can be understood as the mapping from world to body space.
one time step, all populations $f_i$ that “hit” the particle surface during streaming add a particular momentum contribution to the particle. Besides, momentum contributions from the boundary distributions $f^b$—reconstructed according to the moving wall condition from Eq. (5.22)—imply changes to the momentum into the opposite direction. Assuming the wall to be located half-way between the cell centres of neighbouring inner and outer LB cells yields the following expression for the force contributions $F_i$, cf. [103]:

$$F_i(x + \frac{1}{2}c_i dt, t + \frac{1}{2}dt) = c_i(f^*_i(x, t) + f^*_i(x + c_i dt, t))$$

(5.29)

where $x$ denotes the midpoint of the boundary cell, that is the fluid cell neighbouring to the particle geometry, and the velocity vector $c_i$ the lattice velocity pointing towards the particle geometry. The post-collision state $f^*_i(x + c_i dt, t)$ is identical to $f^*_i(x, t + dt)$ from Eq. (5.22). The wall velocity $u_w$ required in the respective equation can be determined from the equations of motion for the particle. More accurate schemes, that is second-order accurate schemes for arbitrary wall locations that are based on interpolation techniques have been developed and proposed in literature, see amongst others [53, 126]. For sufficiently exact geometry representations (which can for example be reached via grid refinement, see Sec. 5.4), the presented method is assumed to deliver satisfactory results; see Chap. 14 and [104] for validation examples. Besides, due to the local availability of all terms that contribute to the forces $F_i$, the current method becomes very attractive. Having determined the force contributions from the fluid that act onto the particle, the particle position and velocity can be updated and the coupling from the flow solver to the particle system is established.

Due to the movement of the particle, the representation of the particle with respect to the LB grid needs to be updated each LB time step. Therefore, the mapping of the particle geometry onto the grid needs to be carried out, converting inner to outer cells and vice versa, cf. Fig. 5.7. The conversion of inner cells to outer cells is straightforward. However, converting outer to inner cells is less trivial as a full set of valid particle distributions needs to be provided for each new inner cell. The following scheme is used to initialise the distributions within a new inner cell at position $x$:

1. Interpolate the density $\rho(x, t)$ using the information from all neighbouring inner cells that have already been inner cells in the previous time step.

2. Obtain the velocity $u(x_{closest}, t)$ where $x_{closest}$ denotes the point on the particle surface with $\|x - x_{closest}\| = \min_{y \in \text{particle surface}} \|x - y\|$.

3. Initialise the particle distributions $f_i := f^{eq}_i(\rho(x, t), u(x_{closest}, t))$.

Some remarks need to be given on this procedure. First, as the creation of a new fluid cell resembles the local creation of additional mass, an initial guess for the density at the respective position is required. In the scheme from above, a simple next-neighbour interpolation is applied. Setting the default density, that is initialising $\rho(x, t) = \rho_L$, would destroy any pressure gradient near the particle surface. Second, due to the low Mach number restriction, both the fluid and the spherical particle move at very low speeds with respect to the discrete time-space ratio $\frac{dx}{dt}$, implying small offsets of the particle positions over subsequent LB time steps. Choosing the velocity from the closest surface point of the particle thus yields a good estimate for the local velocity within the new inner cells. Third, setting local equilibrium for the particle distributions neglects the viscous stresses within the fluid which are determined via the velocity gradients and relate to the non-equilibrium parts $f^{eq}_i$ of the particle distributions. However, again due to the low Mach number restriction, the creation of inner cells within a particular region near the particle surface is a rare event, occurring at an order of $O(10 - 20)$ LB time steps, assuming a unidirectional movement of the particle. Besides, with the new inner cells representing local, singular disturbances, a very fast relaxation and respective adaption of the distributions within these cells takes place. For these reasons, the error introduced by the initialisation method from above is assumed to be small; see [88] for details on this method. As a last note, it shall be remarked that particular boundary treatments become necessary in case that the particle approaches a wall or moves away from
Figure 5.7: Handling of moving geometries in Lattice Boltzmann simulations [88]. Within one Lattice Boltzmann time step, the spherical particle moves from left to right. Its current position and volume is shown in light grey and surrounded by a continuous line whereas its previous location is sketched using a dashed line. The yellow-coloured cells that used to be inner cells are converted into outer cells. In contrast, the blue-coloured cells become new inner cells.

<table>
<thead>
<tr>
<th>Continuum flow</th>
<th>( \text{Kn} &lt; 10^{-2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slip flow</td>
<td>( 10^{-2} &lt; \text{Kn} &lt; 10^{-1} )</td>
</tr>
<tr>
<td>Transition flow</td>
<td>( 10^{-1} &lt; \text{Kn} &lt; 10 )</td>
</tr>
<tr>
<td>Free molecular flow</td>
<td>( 10 &lt; \text{Kn} )</td>
</tr>
</tbody>
</table>

Table 5.1: Characterisation of different flow regimes via the Knudsen number \( \text{Kn} \) [88, 138].

it. In case of the latter, the interpolation scheme from above to initialise new inner cells may not work anymore since no neighbour information might be available for each new cell in this case. Besides, with the particle located very close to the wall, lubrication effects occur within the fluid and need to be incorporated into the coupled particle-fluid model, see for example [104].

5.7 Extension to Finite Knudsen Numbers

Due to its kinetic origin and mesoscopic nature, the Lattice Boltzmann method is very attractive to model micro- and nanoflows. As shown in a later section (Sec. 7.2), the underlying equations can be related to the continuum Navier-Stokes theory and also yield a valid description for the macroscale in the asymptotic limit. This limit is defined by a vanishing Knudsen number

\[
\text{Kn} = \frac{\lambda}{l} \to 0,
\]

relating the mean free path \( \lambda \) of molecules to the characteristic length scale \( l \) of the flow problem. For sufficiently small-scaled gas flow scenarios, the mean free path becomes of similar size as the length \( l \), yielding a non-vanishing Knudsen number. In this case, the continuum assumption consequently does not hold anymore, and new models or modifications to existing models are necessary to still describe the correct flow behaviour. Tab. 5.1 summarises the different flow regimes—continuum, slip, transition and free molecular flow regime—that can be separated from each other via the Knudsen number. In the following, the *finite Knudsen range* denotes the slip and transition regime.

Different methods have been proposed to deal with the flow regimes at non-vanishing Knud-
sen numbers. Higher-order continuum equations such as the Burnett equations, cf. [98], have been developed, incorporating additional Knudsen number-dependent terms. However, it could be shown that these equations are not capable of capturing important flow characteristics, even in simple channel flow setups. An example is given in Fig. 5.8 where a plain channel flow with heated walls is considered. As discussed in different publications, amongst others in [171, 175], both Navier-Stokes and Burnett descriptions fail in even qualitatively capturing the pressure and temperature profiles along the cross-section of the channel. Extended Lattice Boltzmann schemes have turned out to be able to capture the nonlinear characteristics and match respective results from DSMC simulations as reported in [98].

In the following, an extension of the Lattice Boltzmann method to the slip and transition flow regime is described, following the previously published article [138]. Different methods exist to accomplish a respective extension, as discussed in [138]:

“Niu et al. [141] extended the entropic model from Ansumali and Karlin to the finite Knudsen range, introducing a relation between the Knudsen number and the relaxation time and adopting the respective diffuse boundary condition to include velocity slip. Works into similar directions have been published at about the same time by Tang et al. [168]. Sbragaglia and Succi presented a new formulation of kinetic boundary conditions for flows at finite Knudsen numbers in [154]; therefore, they proposed models based on slip, reflection and accommodation coefficients. Toschi and Succi proposed a stochastic handling of finite Knudsen number flows within the context of Lattice Boltzmann simulations in [173]. Virtual wall collisions of the Lattice Boltzmann particles are incorporated into the Lattice Boltzmann model, yielding satisfactory results for flow regimes up to Knudsen numbers \( \sim 30 \). Zhang et al. report a successful qualitative Knudsen minimum prediction in [189]. Their results show good agreement for Knudsen numbers up to \( \sim 0.4 \) and only differ for higher numbers, due to numerical errors induced by the increasing value of the BGK-relaxation-time. In order to suppress artificial slip effects near walls, Verhaeghe et al. [177] propose a MRT-based model, including a particular tuning of the relaxation parameters. Their results show excellent agreement in the slip flow regime, however, they point out deficiencies of the slip flow model for higher Knudsen numbers. A respective extension to the transition flow regime has been developed recently by Li et al. [114]. Another approach to rarefied gas modelling using Lattice Boltzmann methods is reported in [128, 90] where—based on the Hermite projection method—higher-order Lattice Boltzmann models are constructed and yield promising results for both slip and transition flow regime.”

Hence, many attempts to extend Lattice Boltzmann schemes to the finite Knudsen regime have already been made. However, only few works are concerned with realistic setups, cf. [30]. The following methodology is mainly based on the approaches from [114] and [177] and shall be shortly reviewed. Therefore, three steps need to be taken: local viscosity adjustment, extension of the boundary conditions and modification of the collision operator.
With the surface-to-volume ratio significantly increasing in micron-sized scenarios, also the influence of the boundaries such as walls strongly increases: the molecules may hit the walls and as a consequence, the mean free path near the walls is shortened. As the mean free path is linearly related to the dynamic viscosity in case of a hard-sphere gas, the influence of the walls yields changes in the fluid viscosity. Numerous methods exist to account for respective viscosity adjustments and have been considered within this context, cf. [151]. In the following, the Bosanquet-type of expression from [12] is used, due to its simplicity and computational efficiency: an effective viscosity \( \mu_e \) is introduced which depends on the Knudsen number of the flow problem:

\[
\mu_e = \frac{\mu}{1 + a \text{Kn}} \tag{5.31}
\]

where the parameter \( a = 2 \) yields satisfying results over a wide range of Knudsen numbers [129]. The Knudsen number is defined in this case as [114]:

\[
\text{Kn} = \sqrt{\frac{3\pi}{2}} \nu \frac{dt}{dx}. \tag{5.32}
\]

Besides the viscosity adjustment, further modifications are required in terms of boundary conditions. The no-slip condition which is used in classical continuum simulations is not valid anymore in case of rarefied gas flows. Additional slip occurs near the walls and needs to be incorporated into the boundary model; depending on the Knudsen regime, a first- or second-order condition in terms of a Knudsen or mean free path series expansion needs to be applied for the velocities. For flows in the slip regime, the first-order condition for the slip velocity \( u_s \) reads:

\[
\mu_s = \sigma \text{Kn} \left| \frac{\partial u}{\partial n} \right|_{\text{wall}} \tag{5.33}
\]

with the parameter \( \sigma = (2 - \sigma_v)/\sigma_v \) and tangential momentum accommodation coefficient \( \sigma_v \). Throughout the following, \( \sigma = 1 \) is assumed. The condition can be enforced according to [177] in Lattice Boltzmann simulations via a linear combination of standard bounce-back and diffusive reflective (BBDR) boundary conditions:

\[
f_i(x, t + dt) = \beta f_i^{*}(x, t) + (1 - \beta) f_i^D(x, t),
\]

\[
f_i^D(x, t) = \sum_{\text{in}} (c_k - u_{\text{wall}}) n_i f_i^{\text{eq}}. \tag{5.34}
\]

In the equations above, \( n \) denotes the normal direction to the boundary. For channel-like scenarios with an average density \( \rho_{\text{out}} \) at the outlet, the ratio \( \beta \) evolves at

\[
\beta = \frac{3\mu - \text{Kn} \frac{dx}{dt} \rho_{\text{out}}}{3\mu + \text{Kn} \frac{dx}{dt} \rho_{\text{out}}}. \tag{5.35}
\]

Similarly, boundary conditions in transition flow scenarios can be formulated, combining bounce-back and specular reflection conditions [114]. The construction rule for specular reflection is demonstrated in Fig. 5.9. The boundary condition reads:

\[
f_i(x, t + dt) = \beta f_i^{*}(x, t) + (1 - \beta) f_i^S(x, t)
\]

where \( f_i^S \) denotes the specularly reflected distribution. In order to match the second-order slip boundary condition

\[
u_s = A_1 \sigma \lambda_e \left| \frac{\partial u}{\partial n} \right|_{\text{wall}} - A_2 \sigma \lambda_e^2 \left| \frac{\partial^2 u}{\partial n^2} \right|_{\text{wall}} \tag{5.37}
\]

with \( A_1 = (1 - 0.1817\sigma) \), \( A_2 = 0.8 \) and effective mean free path \( \lambda_e = \mu_e/p \cdot \sqrt{\pi RT/2} \), the weighting factor \( \beta \) evolves at

\[
\beta = \frac{1}{1 + A_1 \sigma \sqrt{\frac{x}{6}}}. \tag{5.38}
\]
Figure 5.9: Specular reflection boundary condition [151], assuming the 2D-case and a wall that is placed half-way between outer and inner lattice nodes. Left: distributions bouncing onto the wall. Right: specularly reflected distributions after they collided with the wall.

Finally, a modification in the collision process is required. The BGK collision operator—combined with the half-way bounce-back boundary condition—induces additional slip effects near the walls, depending on the value of the relaxation parameter $\tau$ [68]. In transition and in slip flows, where an exact modelling of the slip at the boundaries is required, this numerical artefact [119] needs to be suppressed. One approach to resolve this issue is the application of the multiple-relaxation-time scheme, similar to the fluctuating collision model from Eq. (5.17) (but without the respective Gaussian noise): the relaxation parameters for the higher-order moments can be tuned depending on the viscous moments and remove the numerical slip at the boundaries [67]. An exact modelling of the slip velocities following the first- and second-order relations from Eqs. (5.33) and (5.37) becomes thus possible. For the relations of the respective relaxation parameters, see [177].

Using the extensions from this section, the Lattice Boltzmann method delivers acceptable descriptions for flows at Knudsen numbers $Kn$ of order $O(1)$. Results for respective flow scenarios are reported in Chap. 15.

5.8 Features and Limitations

Lattice Boltzmann methods are mesoscopic methods, modelling the fluid flow on a statistical level. Although the default Lattice Boltzmann scheme models ideal gases, molecular effects can be coarse-grained and incorporated into Lattice Boltzmann descriptions to a certain extent, allowing for the simulation of non-ideal systems. Respective examples for the latter statement include most multicomponent and multiphase models in the Lattice Boltzmann context, such as the Shan-Chen model [157]. Another example has been shown in Sec. 5.3 in form of the fluctuating collision operator model, resolving Brownian motion effects in the fluid. Brownian modelling is possible on the Navier-Stokes scale as well. However, incorporating strictly local Gaussian noise while preserving stability and allowing for mass and momentum conservation is far from being a trivial task, especially when using incompressible Navier-Stokes solvers. Besides, even for ideal gas simulations, not all molecular states can be captured by the LBM: due to the low Mach number assumption, highly compressible flows cannot be handled by the default LBM (extensions to compressible cases can be found in literature and shall not be considered further at this stage). Besides, rarefied gas flows at very high Knudsen numbers may also not be captured by the LBM so far. Extensions to further approach the rarefied flow regime are subject to current research, cf. Sec. 5.7.

Default Lattice Boltzmann methods are time-explicit and computationally cheap schemes. Only nearest-neighbour communication is required once per time step, all other computations can be carried out locally in each grid cell. This makes the Lattice Boltzmann algorithm attractive for the usage on both shared and distributed memory platforms on the one hand; a particular focus has been put on efficient GPU implementations throughout the last years, see for example [51, 118, 155]. On the other hand, as flow information is only transported one grid spacing per time step, a huge number of time steps is required when solving large-scale setups.

The LBM represents a scheme for weakly compressible flow simulations. This property can
be helpful or problematic, depending on the current problem under investigation. Considering fluid-structure interaction problems, pressure waves that may be generated near the structural boundaries can be damped within their neighbourhood so that subiterations on the fluid solver side can be performed locally, i.e. close to the moving structure [62]. This can reduce the computational effort on the fluid solver side. The local nature of the pressure waves in LBM-based simulations can also turn into a disadvantage: especially when “good” initial conditions for a Lattice Boltzmann simulation are not available or unknown, these waves may be created and first need to decay before a physically reasonable flow field can be obtained. Different approaches have been discussed to tackle this pressure wave problem; for example, the so-called incompressible Lattice Boltzmann model has been proposed [79], taking the density—and hence the pressure—only into account in the zero-order contribution of the equilibrium distribution. This removes compressibility effects of the order $O(Ma^2)$. A complete elimination, however, is not possible.

Compared to the computationally intensive molecular dynamics simulations, cf. Sec. 4.3, the LBM is bound by its memory requirements: in each grid cell, $Q$ distributions need to be stored, and most simulation codes have additional buffers for the density $\rho$ and the flow velocity $u$. Besides, depending on the implementation of the streaming step and required spatial or temporal interpolations, even a second set of distributions may be required. For multicomponent simulations, this problem even becomes worse since a whole set of distributions is stored in this case for each component.

6 Macroscopic Modelling: The Navier-Stokes Equations

Approaching the upper right corner of the scale separation map from Fig. 2.1, the most common description of fluid dynamics is given by the Navier-Stokes equations [162]. They are based on the continuum approach, where each subvolume in space is considered to be homogeneously filled with fluid. Consequently, coarse-grained averaged quantities such as the fluid density, pressure or velocity can be introduced and assumed to be well-defined for any fluid volume of interest. Extending this approach to infinitesimal fluid volumes yields a pointwise description of the fluid in terms of these averaged quantities and allows for the respective formulation of conservation laws in form of partial differential equations. Conservation needs to be guaranteed for the quantities mass, momentum and energy—restricting the discussions and simulation scenarios to isothermal setups, the latter plays a minor role and therefore is not further discussed.

In the following section, the equations are shortly reviewed, and the dimensionless form for the incompressible case is derived; the latter builds the starting point for the numerical schemes to be used throughout the following. The discretisation and the numerical treatment of the Navier-Stokes system are described in Sec. 6.2. Finally, a short summary discusses features and limitations of the arising method.

6.1 Model Description

Extensive derivations of the Navier-Stokes equations can be found in standard text books on computational fluid dynamics, amongst others [44, 52]. A link to statistical mechanics including a respective derivation is discussed in Sec. 7.2. The equations read:

$$\partial_t \rho + \sum_{\alpha} \partial_{x_\alpha} (\rho u_\alpha) = 0$$

(6.39)

$$\partial_t (\rho u_\beta) + \sum_{\alpha} \partial_{x_\alpha} (\rho u_\alpha u_\beta) = -\partial_{x_\beta} p - \sum_{\alpha} \partial_{x_\alpha} \tau_{\alpha\beta} + \rho a_\beta, \ \beta \in \{1, ..., D\}$$

(6.40)

where $\rho$, $u$ and $p$ denote the fluid density, velocity and pressure. Additional volume forces are represented in form of an additional acceleration term $a$. The volume forces can be related to external effects such as gravitational forces. The stress tensor $\tau_{\alpha\beta}$, $\alpha, \beta \in \{1, ..., D\}$, is related to the internal viscous effects of the fluid. In the following, the fluid is assumed to
be Newtonian, i.e. the arising shear stresses are linearly related to the shear velocity. The stress tensor has the form

\[ \tau_{\alpha\beta} := -\mu (\partial_{x\beta} u_\alpha + \partial_{x\alpha} u_\beta) + \frac{2\nu}{3} \delta_{\alpha\beta} \sum_{\gamma} \partial_{x\gamma} u_\gamma \]  

(6.41)

with dynamic viscosity \(\mu\). Due to this simple relation, the set of equations for mass and momentum can be closed by directly inserting the definition from Eq. (6.41) into the momentum equations (6.40). The Newtonian assumption is valid for many fluids of practical interest such as water or air on the macroscale. For Non-Newtonian fluids, separate sets of partial differential equations might be required to define the stress tensor entries \(\tau_{\alpha\beta}\) [52].

Besides, when approaching the nanoscale regime, fluctuating effects—stemming from the Brownian motion of the molecules—need to be incorporated into the fluid stresses [107]. According to Landau and Lifshitz, the arising stress tensor \(\tau_{\alpha\beta}^{\text{mso}} = \tau_{\alpha\beta} + \tau_{\alpha\beta}^{\text{br}}\) consists of the Newtonian stresses and a fluctuating part which has the properties [107, 159].

Equation (6.39) describes the conservation of mass and is known as the continuity equation: the density at a particular position \(x\) changes over time \(\partial_t \rho\) according to the spatial variation of the mass flux at this point and thus needs to balance \(-\sum_\alpha \partial_{x\alpha} (\rho u_\alpha)\). In order to understand the relation, it is worth to consider a short example. Assume that the mass flux is constant inside a particular volume, i.e. \(\partial_{x\alpha} (\rho u_\alpha) = 0\) for all indices \(\alpha\), and the mass is homogeneously distributed in the domain at time \(t = 0\), implying that \(\rho(x,t=0) = \rho_0\). If the same mass enters and leaves the volume under consideration, it is expected that the density \(\rho\) cannot be subject to any changes and must be constant. This simple example can be solved explicitly. Including the constant mass flux assumption in Eq. (6.39) yields \(\partial_t \rho = 0\). The latter equation for \(\rho(x,t)\) can be solved by using the homogeneous initial mass distribution and results in \(\rho(x,t) = \rho_0\) which is consistent with the expected behaviour. A similar interpretation holds for the conservation of momentum in the flow system and is described by Eq. (6.40). However, additional terms are required to be considered in this case: the change in momentum over time \(\partial_t (\rho u_\beta)\) is in balance with the spatial variation of momentum fluxes \(-\sum_\alpha \partial_{x\alpha} (\rho u_\alpha u_\beta)\), surface forces \(-\partial_{x\beta} p\), dissipative forces \(-\sum_\alpha \partial_{x\alpha} \tau_{\alpha\beta}\) and volume forces \(\rho \delta_{\alpha\beta}\).

A simplification of the Navier-Stokes system from above can be derived in the incompressible limit, that is for a fluid with constant density \(\rho\), and builds the starting point for the formulation of the numerical schemes in the subsequent sections. The assumption of incompressibility is valid for flows at low Mach numbers\(^{15} \). With \(\rho = \text{const.}\), the Navier-Stokes

\(^{14}\)The dynamic and kinematic viscosity are related via \(\mu = \nu \cdot \rho\).

\(^{15}\)The Mach number is typically required to be less than 0.3.
system can be written as follows:

\[ \sum_{\alpha} \partial_{x_\alpha} u_\alpha = 0 \]  \hspace{1cm} (6.43)

\[ \partial_t u_\beta + \sum_{\alpha} \partial_{x_\alpha} (u_\alpha u_\beta) = -\frac{1}{\rho} \partial_{x_\beta} p + \nu \sum_{\alpha} \partial_{x_\alpha}^2 u_\alpha + a_\beta, \quad \beta \in \{1, \ldots, D\}. \]  \hspace{1cm} (6.44)

Similar to the Lattice Boltzmann method, a dimensionless form of the equation system to be solved is also preferred in the context of Navier-Stokes simulations. This methodology on the one hand allows a facilitated comparison and scaling between numerical and experimental results and on the other hand may reduce numerical errors due to a more sophisticated scaling of the numerical parameters and unknowns in the equation system. In the case of the incompressible Navier-Stokes system, reference values \( l_{\text{ref}}, u_{\text{ref}}, \rho_{\text{ref}} = \rho \) are introduced for velocity, length and density. A factorisation of all quantities in Eqs. (6.43), (6.44) according to

\[ x_\alpha = l_{\text{ref}} \cdot x'_\alpha \]
\[ u_\alpha = u_{\text{ref}} \cdot u'_\alpha \]
\[ t = \frac{u_{\text{ref}} l_{\text{ref}}}{v} \]
\[ \rho = \rho_{\text{ref}} \cdot \frac{1}{\text{Re}} \]

yields:

\[ \sum_{\alpha} \partial_{x_\alpha} u'_\alpha = 0 \]
\[ \partial_t u'_\beta + \sum_{\alpha} \partial_{x_\alpha} (u'_\alpha u'_\beta) = -\partial_{x_\beta} p' + \frac{1}{\text{Re}} \sum_{\alpha} \partial_{x_\alpha}^2 u'_\alpha + a'_\beta, \quad \beta \in \{1, \ldots, D\}, \]  \hspace{1cm} (6.46)

where \( \text{Re} := \frac{u_{\text{ref}} l_{\text{ref}}}{v} \) is the Reynolds number. Depending on the value of \( \text{Re} \), the flow is considered to be laminar, transient or turbulent. For low Reynolds numbers, as considered within this thesis, laminar flow structures are expected. For a detailed analysis and discussion of the different types of flows, see amongst others [52, 56]. In the rest of Chap. 6, the dimensionless form is to be used exclusively; the \(^{\prime}\) is skipped for the sake of simplicity.

6.2 Discretisation and Numerical Schemes

Numerous discretisation schemes and numerical methods to solve the Navier-Stokes equations can be found in literature. Most common techniques for the discretisation process comprise finite difference, finite volume and finite element techniques. Within the Peano framework which is to be described in Chap. 9, two numerical Navier-Stokes solvers are available: a highly sophisticated finite element-based implementation [133] and a basic finite difference implementation which is based on the descriptions in [70]. In order to simplify couplings of the continuum and sub-continuum scales, the results discussed in Part IV use the finite difference solver. Note, however, that all developed methodologies carry over to other discretisation schemes as well. In the following, the important aspects of the finite difference scheme are shortly reviewed. For extensive explanations, the reader shall be referred to the respective book by Griebel et al. [70].

Numerous approaches to solving the Navier-Stokes equations exist, comprising amongst others solvers for the transport equation of the vorticity, pressure-iteration and pressure-correction schemes as well as methods based on the concept of artificial compressibility [52]. The scheme used herein belongs to the group of pressure-iteration schemes: discretising the time derivative in the momentum equations by a finite difference expression—in the present case, the explicit Euler method—and applying a time-implicit discretisation for the pressure yields the following update rule for the velocities:

\[ u'^{n+1}_{\beta} = u'^{n}_{\beta} + dt \left( -\sum_{\alpha} \partial_{x_\alpha} (u'_\alpha u'_\beta) - \partial_{x_\beta} p'^{n+1} + \frac{1}{\text{Re}} \sum_{\alpha} \partial_{x_\alpha}^2 u'_\alpha + a'_\beta \right), \quad \beta \in \{1, \ldots, D\}. \]  \hspace{1cm} (6.47)
The pressure can be determined from the continuity equation (6.43), enforcing mass conservation at time step \( n + 1 \), that is \( \sum_\alpha \partial_{x_\alpha} u_\alpha^{n+1} = 0 \). This results in a Poisson-like equation for the pressure \( p^{n+1} \):

\[
\sum_\beta \partial_{x_\beta}^2 p^{n+1} = dt \sum_\beta \partial_{x_\beta} \left( - \sum_\alpha \partial_{x_\alpha} (u_\alpha^n u_\beta^n) + \frac{1}{Re} \sum_\alpha \partial_{x_\alpha}^2 u_\alpha^n + a_\beta^n \right).
\] (6.48)

The overall algorithm to solve the incompressible Navier-Stokes equations is sketched in Alg. 6.1. The determination of the new time step size is established such that the CFL condition

Algorithm 6.1 Algorithm to solve the incompressible Navier-Stokes equations.

\[
\begin{align*}
t &= t_{\text{start}} \\
\text{while } & t < t_{\text{end}} \text{ do} \\
& \quad \text{determine new time step size } dt \\
& \quad \text{assemble right hand side of pressure Poisson equation (6.48)} \\
& \quad // \; \text{This yields the pressure in the next time step} \\
& \quad \text{solve pressure Poisson equation for } p \\
& \quad // \; \text{This yields the new velocity in the next time step} \\
& \quad \text{compute velocity updates } u \text{ according to Eq. (6.47)} \\
& \quad t = t + dt \\
\text{end while}
\end{align*}
\]

and time step restrictions due to the diffusive Navier-Stokes terms are fulfilled during this time step [70]. This guarantees numerical stability of the scheme.

The unknowns of the system—velocities and pressure values—can be mapped onto a Cartesian grid in a staggered manner, cf. Fig. 6.1 on the left, to enhance stability. The momentum equations are discretised at the grid points where the respective entries of the velocity vectors are placed, i.e. the momentum equation for the velocity direction \( u_0 \) is evaluated in the midpoint of the right edge of the grid cells and the equation for \( u_1 \) is solved in the midpoint of the upper edge of the cells. The pressure Possion equation which enforces mass conservation is approximated in the cell midpoints, that is directly at the positions where the pressure unknowns are stored.

The partial spatial derivatives for the pressure gradient, convective and diffusive terms are approximated by a weighted form of central and upwind differences. The latter is applied to allow for a stable, but first-order solution in convection-dominated problems whereas central differences are used to obtain a close-to-second-order accurate solution in the diffusive case. The evaluation of the non-linear term \( \partial_{x_1}(u_0 u_1) \) in terms of central differences including the respective interpolation weights is exemplarily illustrated in Fig. 6.1 on the right.

6.3 Particle-in-Flow Simulations Based on Faxén’s Theorems

In Sec. 5.6, a two-way coupling approach for the simulation of particles which are suspended in flows has been discussed. This approach is generally valid for arbitrary particle forms and flow scenarios. However, due to the two-way coupling and the explicit representation of the particle on the Cartesian grid, a fine grid and consequently high computational resources are required to solve these kinds of particulate flow setups. A simplification of the particle model for respective flow scenarios of interest might therefore be desirable.

One approach to model the particle-in-flow problem is given by Faxén’s theorems [48] which were derived by Hilding Faxén in 1921 after previous works by Stokes, Oseen and Lamb. A historical overview on the respective works can for example be found in [117]. Given a velocity field for a Stokes flow problem, i.e. for a steady-state non-convective flow (Re \( \ll 1 \)), a particle is virtually placed inside the flow. The particle can hence feel accelerations from the fluid, but it does not have any impact on the flow field, resembling a one-way coupling
of the particle to the fluid flow. Faxén computed the force and torque acting on a (three-dimensional) spherical particle which is located far away from any walls:

\[
F_p = 6\pi\mu r (u - v_p) + \pi\mu r^3 \sum_\alpha \partial_\alpha^2 u \\
L_p = 4\pi\mu r^3 (\nabla \times u - 2\omega_p).
\] (6.49) (6.50)

In this equation, \(r\) denotes the radius of the sphere and \(u\) is the fluid velocity at the centre of the spherical particle. An analogous expression for the translational forces in two-dimensional simulations has been derived within the scope of the DiParTS-project and a master’s thesis [92]. The derivatives in Eqs. (6.49), (6.50) can for example be evaluated using finite differences, including a respective interpolation of the required velocity values at the particle’s centre. A second-order interpolation is consistent with the finite difference scheme for the Navier-Stokes equations from Sec. 6.2 and hence provides a sufficient level of accuracy.

In order to compute the trajectory of the particle, Alg. 6.2 can be applied. The particle can hence be simulated in a simple post-processing step, cf. line 5 of the algorithm. Results using the Faxén approach together with the finite difference Navier-Stokes solver in Peano are presented in Part IV.
Table 7.1: Features and limitations of the different methods under consideration.

<table>
<thead>
<tr>
<th></th>
<th>Molecular dynamics</th>
<th>Lattice Boltzmann methods</th>
<th>Incompressible Navier-Stokes solvers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level of detail</td>
<td>Molecular level</td>
<td>Statistical level</td>
<td>Continuum level</td>
</tr>
<tr>
<td>Knudsen range</td>
<td>Everywhere</td>
<td>Continuum – transition</td>
<td>Continuum – slip</td>
</tr>
<tr>
<td>regime</td>
<td></td>
<td>regime</td>
<td></td>
</tr>
<tr>
<td>Finest spatial resolution</td>
<td>Molecular length</td>
<td>Nano-sized length scale</td>
<td>Micro-/ Nano-sized length scale</td>
</tr>
<tr>
<td>scale</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coarsest spatial resolution</td>
<td>$O(\mu m)$</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Level of compressibility</td>
<td>Compressible</td>
<td>Weakly compressible</td>
<td>Incompressible</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time stepping</td>
<td>Explicit</td>
<td>Explicit</td>
<td>Explicit/ implicit</td>
</tr>
<tr>
<td>Computational load per time step</td>
<td>Medium</td>
<td>Small</td>
<td>High</td>
</tr>
<tr>
<td>Memory requirements</td>
<td>Small</td>
<td>High</td>
<td>Small</td>
</tr>
<tr>
<td>Scalability</td>
<td>Good</td>
<td>Good</td>
<td>Medium</td>
</tr>
</tbody>
</table>

6.4 Features and Limitations

Compared to Lattice Boltzmann methods, the memory requirements of Navier-Stokes solvers are very low with $1 + D$ unknowns per grid cell for pressure and velocity values. Instead, they are computation-bound: the incompressibility assumption implies that the fluid has an infinite speed of sound, yielding an infinitely fast propagation of pressure waves. For the numerical scheme from Sec. 6.2, this results in a Poisson equation for the pressure that needs to be solved in every time step and hence implies high computational costs. Highly efficient parallel linear solvers are consequently required in case of large-scale flow scenarios, ranging from relaxation-based over preconditioned CG- and GMRES-based solvers to multigrid schemes. However, several examples—such as flows in highly complex geometries—exist where the numerical realisation of an efficient solver may become a non-trivial problem.

Besides the presented explicit Euler-based time integration scheme which requires small time steps due to stability restrictions, time-implicit schemes exist and allow for significantly larger time step sizes. This consequently yields a more efficient treatment of steady-state and other laminar flow problems. In this case, however, it is even a non-linear system of equations that needs to be solved in each time step.

7 From Micro to Macro: Scale Transitions

The features and limitations of the different methods, that is molecular dynamics, Lattice Boltzmann methods and incompressible Navier-Stokes solvers (cf. Sec. 4.3, 5.8 and 6.4), are listed in Tab. 7.1. This table is not complete, and to a certain extent, it also represents the author’s point of view. Still, several conclusions can be drawn from this overview. First, a spatial resolution of the molecular length scale is hard to reach with Lattice Boltzmann schemes and not possible for Navier-Stokes solvers. Although groups apply very fine grids with sub-nano-sized spatial resolutions for Lattice Boltzmann simulations (see also Sec. 13.1), the validity of the arising simulation scheme is very questionable. Second, due to increasing computational costs, the molecular dynamics schemes are not suited to simulate domains which exceed the micrometer range. Third, all schemes come with different computational loads per simulation time step and with different memory requirements. Fourth, bigger time steps can be achieved for implicit Navier-Stokes solvers compared to the (standard) explicit Lattice Boltzmann schemes.

All these points imply that there is no “unique best fluid solver” for general applications in
computational fluid dynamics. Every method comes with particular features and limitations—and in some cases, a fluid solver’s feature represents another solver’s limitation. For example, molecular dynamics resolves molecular structures but is computationally too expensive for bigger-sized scenarios. In contrast, Lattice Boltzmann methods cannot resolve single molecules and can be applied to solve fluid flow on much bigger time and length scales. Hence, combining the two approaches into a unified scheme may result in a hybrid method that provides both resolving bigger time and length scales as well as molecular structures. Before constructing such hybrid methods, the theory that connects the three fluid flow descriptions shall be reviewed. The respective relations are considered to be of essential importance to understand and develop the hybrid schemes, such as coupled molecular dynamics–Lattice Boltzmann or Lattice Boltzmann–Navier-Stokes simulations, cf. Chap. 11 and 13. The link between molecular and mesoscopic Lattice Boltzmann descriptions is discussed in Sec. 7.1. Therefore, the molecular description is converted into the Boltzmann equation. The Boltzmann equation is subsequently discretised and yields the Lattice Boltzmann description. The relation between the Lattice Boltzmann method and the Navier-Stokes equations is laid out in Sec. 7.2. For a detailed analysis of the transition from the Boltzmann equation to Lattice Boltzmann methods, see amongst others [80, 158]. A more comprehensive description of the way from Lattice Boltzmann to Navier-Stokes can be found in [24, 108] or various papers on the Chapman-Enskog expansion technique such as [42].

### 7.1 From Molecules to Populations

The starting point of the following analysis builds the idea of the equations of motion, described before in Sec. 4.1:

\[
\frac{\mathrm{d}x_p}{\mathrm{d}t} = v_p \\
\frac{\mathrm{d}v_p}{\mathrm{d}t} = \frac{1}{m_p} F_p.
\]  

(4.1)

Similar to the movement of a single molecule in space, one can also think of tracking a distribution density of particles. This density \( g \) lives in the phase space and thus depends on the local position in space \( x \), time \( t \) and the microscopic fluid velocity \( v \). In other words, \( g = g(t, x, v) \) describes the probability density to find molecules at a spatial location \( x \) at time \( t \) that move with a velocity \( v \). Molecules can only change their velocity due to interactions with other molecules, analogously to the arising forcing terms \( F_p \) in Eq. (4.1). This also needs to hold for the probability density. Hence, a change of the density \( g = g(t, x(t), v(t)) \) along a particular trajectory over time may only be induced via intermolecular collisions:

\[
\frac{\mathrm{d}g}{\mathrm{d}t} = \Delta(g - g^{eq})
\]  

(7.51)

where \( \Delta(g - g^{eq}) \) represents a coarse-grained model for the probabilistic influence of intermolecular collisions. The term \( g^{eq} \) defines the equilibrium state of the system under consideration; for a given average velocity \( u(x, t) \), it is defined via the following Maxwellian distribution [163]:

\[
g^{eq}(t, x, v) = \rho(x, t) \left( \frac{m_p}{2\pi k_BT} \right)^{D/2} \exp \left( -\frac{m_p(v - u(x,t))^2}{2k_BT} \right). \]  

(7.52)

The equilibrium distribution can be derived via Boltzmann’s H-theorem under the assumptions of binary uncorrelated collisions between particles. It has been shown (and is stated in the second law of thermodynamics), that the entropy function, given by \( H(t) := -\int g(t, x, v) \ln g(t, x, v) \, \mathrm{d}x \, \mathrm{d}v \), may only increase over time, that is \( \frac{\mathrm{d}H}{\mathrm{d}t} \geq 0 \), for any solution of \( g \) of the Boltzmann equation. In this statement, equality is fulfilled if and only if \( g \) is a Maxwellian; for the proof, see amongst others [182]. Transforming the ordinary differential
equation description from Eq. (7.51) using \( \frac{\partial}{\partial t} = \partial_t + \frac{\partial x}{\partial t} \cdot \nabla_x + \frac{\partial v}{\partial t} \cdot \nabla_v \) with \( v = \frac{\partial x}{\partial t} \) and neglecting any accelerating forces\(^{18}\) results in the Boltzmann equation:

\[
\partial_t g + \mathbf{v} \cdot \nabla_x g = \Delta(g - g^{eq}).
\] (7.53)

In order to step towards the Lattice Boltzmann equation, the velocity space needs to be discretised. A straightforward approach to this task would be the application of finite difference or finite element schemes. However, with the distribution \( g(t, \mathbf{x}, \mathbf{v}) \) depending on 2D+1 coordinates and assuming a similar technique for the discretisation of time and space, it becomes obvious that the arising discrete system grows extremely fast. Having 10 discrete points for each coordinate axis would already yield a set of 10\(^6\) points for a three-dimensional flow problem at a single time step! The respective spatial resolution of 10 \times 10 \times 10 points may only allow for the simulation of very simple scenarios. Therefore, a different approach to velocity space discretisation is required. First, the considered fluid is assumed to only be slightly compressible, implying that the mean velocity of the fluid. This assumption allows a series expansion of the equilibrium state from Eq. (7.52) in terms of low velocities \( \mathbf{v} \). Introducing the speed of sound \( c_s := \sqrt{k_B T/m_p} \) yields the following approximation to the equilibrium distribution:

\[
g^{eq}_h(t, \mathbf{x}, \mathbf{v}) = \rho(\mathbf{x}, t) \left( \frac{m_p}{2\pi k_B T} \right)^{D/2} \exp\left( -\frac{\mathbf{v}^2}{2c_s^2} \right) \left( 1 + \frac{(\mathbf{v} \cdot \mathbf{u})^2}{2c_s^2} - \frac{\mathbf{u}^2}{2c_s^2} \right)
\] (7.54)

with \( g^{eq} = g^{eq}_h + O(\mathbf{u}^2) \). It can be observed that this approximation is already close to the discrete representation of the equilibrium distribution \( f_i^{eq} \) from Eq. (5.10). Second, a suitable set of discrete velocities \( c_i, i = 1, ..., Q \), needs to be chosen. In order to obtain the Lattice Boltzmann description from Chap. 5, this choice has to fulfill the following conditions \([80]\):

C1 The velocity vectors shall have the correct length and orientation, such that the discrete molecular movement can be mapped onto the Cartesian grid. In other words, one time step shall correspond to the movement of the molecules from one cell to its neighbours.

C2 The hydrodynamic moments of the system need to be recovered from an integration over the velocity space, similar to a summation or averaging process over all molecules. For example, the mass of the total system shall result from the summation over all molecules. Hence, a velocity discretisation is considered to be valid, as long as the macroscopic hydrodynamics are still satisfied (C2) and the fluid transport can be carried out on a (typically simple) grid structure (C1). A detailed discussion of the conditions C1 and C2, together with the derivation of the D2Q9 velocity set, the respective quadrature rule and its lattice weights \( w_i \) can be found in \([80]\). Analogously, other discretisation schemes such as D3Q15, D3Q19 or D3Q27 can be analysed and derived. Tab. 7.2 lists different velocity discretisation schemes and their lattice weights. An alternative route towards the derivation of the lattice weights can be found in \([158]\) where Shan and He apply a Hermite polynomial expansion to the distribution \( g \). From the minimum requirements C1 and C2, the velocity space could be reduced to a finite small set of lattice velocities \( c_i, i = 1, ..., Q \). For the Boltzmann equation, this implies that its discrete representation (with respect to the velocity space discretisation) is given by a set of \( Q \) equations:

\[
\partial_t g_i^h(\mathbf{x}, t) + c_i \cdot \nabla_x g_i^h(\mathbf{x}, t) = \Delta_i(g_i^h(\mathbf{x}, t) - g_i^{eq}(\mathbf{x}, t)), \quad i = 1, ..., Q
\] (7.55)

\(^{18}\)Neglecting the influence of accelerating forces implies that the term \( \frac{\partial}{\partial t} \cdot \nabla_v g = \frac{1}{m_p} \mathbf{F} \cdot \nabla_v g \approx 0 \) where \( \mathbf{F} \) represents the statistical analogue to the particle force \( \mathbf{F}_p \).
Table 7.2: Velocity sets and lattice weights for different velocity discretisations in two and three dimensions. The column “Velocities” denotes the different speeds that the populations are assigned to, assuming an underlying Cartesian grid with unit mesh size ($dx = 1$).

\[
\begin{array}{|c|c|c|}
\hline
\text{Velocity Set} & \text{Velocities} & \text{Lattice Weights } w_i \\
\hline
\text{D2Q9} & 0 & \frac{4}{9} \\
 & 1 & \frac{1}{9} \\
 & \sqrt{2} & \frac{1}{36} \\
\hline
\text{D3Q15} & 0 & \frac{2}{9} \\
 & 1 & \frac{1}{9} \\
 & \sqrt{3} & \frac{1}{72} \\
\hline
\text{D3Q19} & 0 & \frac{1}{3} \\
 & 1 & \frac{1}{18} \\
 & \sqrt{2} & \frac{1}{36} \\
\hline
\text{D3Q27} & 0 & \frac{8}{27} \\
 & 1 & \frac{2}{27} \\
 & \sqrt{2} & \frac{1}{54} \\
 & \sqrt{3} & \frac{1}{216} \\
\hline
\end{array}
\]

where $g_h^i(x, t)$ represents the discretised counterpart of $g(t, x, c_i)$. The distribution $g_{h}^{eq}(x, t)$ denotes the vector of respective equilibrium states $g_{h}^{eq}(t, x, c_i)$ that are based on Eq. (7.54) and the derived lattice weights $\star \star \star$.

The spatial gradient is discretised along the trajectory of the lattice velocity $c_i$ and is evaluated at time $t + dt$. Choosing the step width along the trajectory correspondingly as $dx_i := c_i \cdot dt$ yields:

\[
c_i \cdot \nabla_x g_i^h \approx \frac{\|c_i\| \cdot g_i^h(x+dx_i, t+dt) - g_i^h(x, t+dt)}{dx_i} = g_i^h(x+dx_i, t+dt) - g_i^h(x, t+dt). \tag{7.57}
\]

Setting the discrete approximations from Eq. (7.56) and (7.57) into Eq. (7.55) results into the Lattice Boltzmann equation:

\[
g_i^h(x + c_i dt, t + dt) = g_i^h(x, t) + dt \cdot \Delta_i(g_i^h - g_{h}^{eq}). \tag{7.58}
\]
order to obtain a second-order accurate update scheme, a discretisation of \( \Delta(g_h - g_{eq}^h) \) is required, making use of the Crank-Nicolson scheme. In order to retain the explicit nature of the time stepping scheme, a redefinition of the distribution functions as well as a shift in the linear relationship between relaxation times and the kinematic viscosity is required. The overall (matrix) form of the collision operator remains though. For details on the derivation of the discrete collision operator form, see amongst others [37, 78].

7.2 From Populations to Macroscopic Conservation Laws

In the previous section, the molecular dynamics approach has been set into connection with the mesoscopic Lattice Boltzmann description. Next, the relation between the Lattice Boltzmann method and the continuum description of fluid dynamics in terms of the Navier-Stokes equations shall be reviewed. The respective analysis is carried out analogously to the author’s publication [134]. It is based on the Chapman-Enskog expansion technique [24]. In the following, the major steps of this technique are described to highlight both the fundamentals of the expansion and the underlying theory for the coupling of Lattice Boltzmann and Navier-Stokes solvers from Sec. 16.2. Details on the method can be found amongst others in [24, 163, 182].

In order to connect the meso- and macroscales, the Lattice Boltzmann equation (5.8) is considered in the asymptotic continuum limit, i.e. for

\[ \epsilon_{Kn} := \frac{l}{l_H} \to 0. \] (7.59)

The parameter \( \epsilon_{Kn} \) is similar to the Knudsen number and is therefore denoted by the subscript Kn. In Eq. (7.59), \( l \) and \( l_H \) denote the characteristic length scales of the Lattice Boltzmann and the hydrodynamic (macroscopic) system. Based on the expansion parameter \( \epsilon_{Kn} \), the spatial coordinate \( x_H \) of the macroscopic description can be related to its mesoscopic counterpart, which is used by the Lattice Boltzmann method, as follows:

\[ x_H := \epsilon_{Kn} x. \] (7.60)

A similar relation can be formulated for the time scales of the macro- and the mesoscopic description. However, in order to capture both convective and diffusive phenomena, a separation of the respective effects needs to be established. Diffusion effects happen very fast and are related to the diffusive time scale \( t_D \) whereas the slower convective behaviour is taking place on the longer time scale \( t_C \). Both time scales \( t_C \) and \( t_D \) can be defined as:

\[ t_C := \epsilon_{Kn} t \]
\[ t_D := \epsilon_{Kn}^2 t. \] (7.61)

The asymptotic analysis of the Lattice Boltzmann equation starts with a Taylor expansion of the right hand side of Eq. (5.8) at \( (x, t) \):

\[
\begin{align*}
f_i(x + c_i dt, t + dt) &= f_i(x, t) \\
&+ \sum_{\alpha} c_{i\alpha} dt \partial_{x_{\alpha}} f_i(x, t) + dt \partial_t f_i(x, t) \\
&+ \frac{dt^2}{2} \sum_{\alpha, \beta} c_{i\alpha} c_{i\beta} \partial_{x_{\alpha}} \partial_{x_{\beta}} f_i(x, t) + dt^2 \sum_{\alpha} \partial_{x_{\alpha}} \partial_t f_i(x, t) \\
&+ \frac{dt^2}{2} \partial_t^2 f_i(x, t) + O(dt^3).
\end{align*}
\] (7.62)

The distribution functions \( f_i \) can be expanded into an asymptotic series near its equilibrium state:

\[
f_i = f_i^{eq} + \epsilon_{Kn} f_i^{(1)} + O(\epsilon_{Kn}^2) \] (7.63)

The shift can be seen in Eq. (5.14) where 0.5 needs to be subtracted from the relaxation time.
where the non-equilibrium part resembles \( f_{i\text{neq}} = \epsilon_{Kn} f_i^{(1)} \) in a first-order expansion. Both density and momentum can be determined from equilibrium and non-equilibrium states, that is

\[
\sum_i f_i = \sum_i f_i'^{\text{eq}} \tag{7.64}
\]
\[
\sum_i f_i c_{i\alpha} = \sum_i f_i'^{\text{eq}} c_{i\alpha}, \quad \alpha \in \{1, ..., D\}. \tag{7.65}
\]

For the non-equilibrium parts \( f_{i\text{neq}} = f_i - f_i'^{\text{eq}} \), it follows:

\[
\sum_i f_i'^{\text{neq}} = 0 \tag{7.66}
\]
\[
\sum_i f_i'^{\text{neq}} c_{i\alpha} = 0, \quad \alpha \in \{1, ..., D\}. \tag{7.66}
\]

The right hand side of Eq. (5.8) can be expanded analogously to the equilibrium expansion from Eq. (7.63):

\[
f_i + \Delta_i (f - f'^{\text{eq}}) = f_i'^{\text{eq}} + \Delta_i^{(0)} (f - f'^{\text{eq}}) + \epsilon_{Kn} \left( f_i^{(1)} + \Delta_i^{(1)} (f - f'^{\text{eq}}) \right) + \epsilon_{Kn}^2 \Delta_i^{(2)} (f - f'^{\text{eq}}) + O(\epsilon_{Kn}^3). \tag{7.67}
\]

Next, the distribution function \( f_i(x, t) \) is redefined using the introduced macroscopic space and time coordinates \( x_H, t_C \) and \( t_D \). Therefore, a new distribution \( \tilde{f}_i \) is introduced:

\[
f_i(x, t) = \tilde{f}_i(x_H(x), t_C(t), t_D(t)). \tag{7.68}
\]

For the partial derivatives, the chain rule delivers:

\[
\partial_t = \epsilon_{Kn} \partial_{t_C} + \epsilon_{Kn}^2 \partial_{t_D} \tag{7.69}
\]
\[
\partial_{x_{H\alpha}} = \epsilon_{Kn} \partial_{x_{H\alpha}}, \quad \alpha \in \{1, ..., D\}. \tag{7.69}
\]

Setting the coordinate transformation from Eqs. (7.68),(7.69) and the equilibrium expansion from Eq. (7.63) into Eq. (7.62) yields:

\[
f_i(x + c_i dt, t + dt) = \tilde{f}_i^{\text{eq}}
\]
\[
+ \epsilon_{Kn} \left( \tilde{f}_i^{(1)} + \sum_{\alpha} c_{i\alpha} dt \partial_{x_{H\alpha}} \tilde{f}_i^{\text{eq}} + dt \partial_{t_C} \tilde{f}_i^{\text{eq}} \right)
\]
\[
+ \epsilon_{Kn}^2 \left( \sum_{\alpha} c_{i\alpha} dt \partial_{x_{H\alpha}} \tilde{f}_i^{(1)} + dt \partial_{t_D} \tilde{f}_i^{\text{eq}} + dt \partial_{t_C} \tilde{f}_i^{(1)} + \frac{d^2}{dt^2} \partial_{t_C} \tilde{f}_i^{\text{eq}} \right)
\]
\[
+ \epsilon_{Kn}^2 \sum_{\alpha} c_{i\alpha} dt \partial_{x_{H\alpha}} \tilde{f}_i^{(1)} + \frac{d^2}{dt^2} \sum_{\alpha} c_{i\alpha} \partial_{x_{H\alpha}} \partial_{t_C} \tilde{f}_i^{\text{eq}} + O(\epsilon_{Kn}^3). \tag{7.70}
\]

Asymptotic theory delivers that terms of same order in \( \epsilon_{Kn} \) from the Eqs. (7.67) and (7.70) need to be equal. For the zero-, first- and second-order, this implies that

\[
\Delta_i^{(0)} = 0 \tag{7.71}
\]
\[
\left( \partial_{t_C} + \sum_{\alpha} c_{i\alpha} \partial_{x_{H\alpha}} \right) \tilde{f}_i^{\text{eq}} = 1 dt \Delta_i^{(1)} (f - f'^{\text{eq}}) \tag{7.72}
\]
\[
\partial_{t_D} \tilde{f}_i^{\text{eq}} + \frac{1}{2} \left( \partial_{t_C} + \sum_{\alpha} c_{i\alpha} \partial_{x_{H\alpha}} \right) \left( \tilde{f}_i^{(1)} + \Delta_i^{(1)} \right) = \frac{1}{dt} \Delta_i^{(2)} (f - f'^{\text{eq}}). \tag{7.73}
\]
Equation (7.71) shows that intermolecular collisions do not alter the system at zero-order.

From Eqs. (7.72) and (7.73), the conservation laws for mass and momentum can be obtained. For this purpose, the equations are multiplied by the factors $1$ and $c_i^\beta$, $\beta = 1, ..., D$, respectively, and are integrated over the velocity space. The latter corresponds to a summation over all lattice velocity directions. In terms of mass conservation, the multiplication of Eqs. (7.72) and (7.73) by $1$ and integration over the velocity space yields:

$$
\partial_t \rho + \sum_\alpha \partial_{x_\alpha} \left( \rho \tilde{u}_\alpha \right) = 0 \tag{7.74}
$$

Following the same procedure, equations for momentum conservation can be obtained:

$$
\partial_t \left( \rho \tilde{u}_\beta \right) + \sum_\alpha \partial_{x_\alpha} \left( 2 f_i^{eq} c_i^\alpha c_i^\beta + \Delta_i^{(1)} \right) = 0 \tag{7.77}
$$

In order to obtain the Navier-Stokes system, the Eqs. (7.74), (7.75) and (7.76), (7.77) are merged via linear combinations of the form $\epsilon \kappa_n \cdot$ Eq. (7.74) $+ \epsilon^2 \kappa_n \cdot$ Eq. (7.75) and $\epsilon \kappa_n \cdot$ Eq. (7.76) $+ \epsilon^2 \kappa_n \cdot$ Eq. (7.77), respectively. This combination, together with the resubstitution of the original variables $x$ and $t$ for space and time results in:

$$
\partial_t \rho + \sum_\alpha \partial_{x_\alpha} \left( \rho \tilde{u}_\alpha \right) = 0
$$

$$
\partial_t \left( \rho \tilde{u}_\beta \right) + \sum_\alpha \partial_{x_\alpha} \left( \rho \tilde{u}_\alpha \tilde{u}_\beta \right) = -\partial_{x_\beta} p + \sum_\alpha \partial_{x_\alpha} \tau_{\alpha\beta} \tag{7.78}
$$

Besides the integration formulas for mass and momentum from Eqs. (7.64) and (7.65), the derivations from the last step further make use of the equalities $\sum_i f_i^{eq} c_i^\alpha c_i^\beta = p \delta_{\alpha\beta} + \rho u_\alpha u_\beta$ for the Eulerian stress and the ideal equation of state $p = c_s^2 \rho$ for the pressure.

In order to match the viscous stresses, it can be further shown that the contributions from the non-equilibrium terms need to satisfy

$$
-\frac{1}{2} \sum_i \left( 2 f_i^{eq} + \Delta_i^{(eq)} \right) c_i^\alpha c_i^\beta = \mu \left( \partial_{x_\beta} u_\alpha + \partial_{x_\alpha} u_\beta \right) + O(u^3) \tag{7.79}
$$

Further analysis is required to prove the correctness of the latter equation, including the determination of the exact relation between the collision operator, i.e. the relaxation times, and the viscosity of the fluid. For the BGK collision model, the relation is given by Eq. (5.14), see for example [134, 161]. Similar expressions can be found for MRT-based collision models. Details on the respective analysis can be found in [24, 42, 76].
Part III

Algorithms and Implementations

In the following, the simulation software which is used within the scope of this PhD thesis is reviewed, and new algorithmic concepts as well as corresponding implementations are described. The part starts with Chap. 8 on molecular dynamics simulations. Two implementations are presented therein: MarDyn, a flexible framework for massively parallel molecular dynamics simulations and a basic molecular dynamics simulation which is integrated into the macro-micro-coupling tool (cf. Sec. 13.4) and which is mostly used for validation purposes. A short introduction to the Peano framework is given in Chap. 9 since the framework builds a common base for the Navier-Stokes and Lattice Boltzmann implementations that are used/developed in this thesis. The Lattice Boltzmann application of Peano builds one main pillar of this part. Its implementation and design is described in-depth in Chap. 10. After a short motivation, efficiency and compatibility aspects of Lattice Boltzmann implementations with respect to the general Peano concept are discussed in Sec. 10.2. The block-structured Lattice Boltzmann scheme arising from this discussion is subsequently explained in Sec. 10.3 and 10.4. Within this context, a new pattern for the streaming step is presented which naturally evolves from the local cell-handling concept of Peano and the need for memory reduction for Lattice Boltzmann schemes. The extension of the Lattice Boltzmann scheme to the spatially adaptive spacetree grid is presented in Sec. 10.5. Particular focus is put onto the development of a new dynamic grid refinement technique for the Lattice Boltzmann scheme. After a brief introduction to the Navier-Stokes solver in Peano (see Chap. 11), the coupling of Navier-Stokes and Lattice Boltzmann solvers is addressed in Chap. 12. A new optimisation-based strategy for a respective macro-to-meso-coupling is introduced in Sec. 12.2 which—based on the Chapman-Enskog analysis from Sec. 7.2—conserves mass, momentum and viscous stresses at the interface between the two solvers. With this new strategy available, the implementation of two Lattice Boltzmann–Navier-Stokes hybrids is described in Sec. 12.3. The part closes with Chap. 13 on hybrid Molecular Dynamics–Lattice Boltzmann simulations. After reviewing modelling and algorithmic aspects for respective spatial coupling schemes in Sec. 13.2, a prototype implementation which couples MarDyn and the spatially adaptive Lattice Boltzmann application of the Peano framework is described in Sec. 13.3. Based on the analysis of the prototype, the macro-micro-coupling tool is developed which is meant to support developers of new molecular–continuum schemes on the one hand and which shall simplify the setup of massively parallel hybrid simulations on the other hand. Its software design and extensions for massively parallel hybrid simulations are discussed in Sec. 13.4.

8 Molecular Dynamics Simulations

Within the scope of this thesis, two molecular dynamics codes are used. The molecular dynamics framework MarDyn [19, 20] is used to identify and analyse the components in coupled molecular–continuum simulations (cf. Sec. 13.3). With a sophisticated molecular dynamics code applied in this analysis, the formulation of the interfaces between the continuum and the molecular solvers is expected to be more general and flexible. The implementation of
the framework is shortly described in Sec. 8.1. Besides, in order to support the developers of new coupling schemes, the macro-micro-coupling tool which is to be developed in Sec. 13.4 comes with a simple built-in molecular dynamics solver. The implementation of this molecular dynamics simulation is explained in Sec. 8.2.

8.1 MarDyn

MarDyn [19, 20] is a framework for three-dimensional massively parallel molecular dynamics simulations. It is developed in a cooperation of chemical engineers and computer scientists since 2005. Its major field of application lies in the study of thermodynamic phenomena. This implies the simulation of big systems of typically small-sized molecules, in contrast to many biological processes which rather involve the simulation of polymers or other macromolecules. A modular design of the framework allows for a simple exchange of molecule models, data structures for the linked cell-based molecule traversal or the underlying parallelisation, that is domain decomposition, strategies. A simplified class diagram of MarDyn is depicted in Fig. 8.1.

The exchange mechanism of the underlying parallelisation strategies represents a specific feature of MarDyn. Based on the feature, the framework also supports various dynamic load balancing strategies which are advantageous in simulations where clusterings of molecules may occur.

MarDyn supports single- and multi-centred molecule representations together with different pairwise interaction potentials such as the Lennard-Jones [147] or the Tersoff potential [170]. The evaluation of the pairwise potentials is implemented according to the adapter design pattern: the ParticleContainer (cf. Fig. 8.1) holds an instance of the PairHandler which holds instances of all relevant pairwise interaction models. The method traversePairs() of the ParticleContainer loops over all pairs and delegates the force evaluation to the PairHandler. For this purpose, the method processPair(molecule1, molecule2) of the PairHandler is triggered for every pair of molecules.

With one major focus consisting in large-scale molecular dynamics simulations, further improvements of MarDyn with respect to both memory and runtime efficiency are subject of current research. A memory reduction technique has recently been published [47], and a vectorised version is under current development.

8.2 Built-in Molecular Dynamics

The design of the molecular dynamics simulation which is included in the coupling tool is depicted in Fig. 8.2. It can be used to simulate single-centred Lennard-Jones fluids in two and three dimensions. Distributed parallel simulations based on MPI are supported.

The molecules are stored in the MoleculeService. This singleton provides access to the...
molecules via a callback-concept. A molecule mapping \texttt{myMapping} implements the methods \texttt{beginIteration()}, \texttt{endIteration()} and \texttt{handleMolecule(Molecule&).} In the \texttt{MolecularDynamicsSimulation}, an object of the mapping is initialised and \texttt{MoleculeService::getInstance().iterate(myMapping)} is called. This triggers an iteration over all molecules on the current process and calls \texttt{handleMolecule(Molecule&)} on each molecule. Similarly, the access to the linked cells is established. The \texttt{LinkedCellService} together with \textit{linked cell mappings} is used to iterate over the molecules in a cellwise manner. Here, the access to a single cell or to a cell and each of its direct neighbours is provided. Different molecule and linked cell mappings are shown in Fig. 8.2. For example, the time integration is carried out using a molecule mapping whereas the Lennard-Jones-based force evaluation is established via a linked cell mapping. The \texttt{ParallelTopologyService} handles the communication between processes in MPI-based parallel simulations. In the parallel case, the domain is split using a regular domain decomposition which is applied to the linked cell structure. Each process thus only handles the molecules inside its local linked cells. In order to allow for simulations in more complex geometrical setups than purely periodic simulations, the \texttt{MolecularGeometryService} can be used. This service defines whether a molecule is freely moving or whether it is fixed in space. Besides, a particular treatment of the boundary regions is necessary. For periodic simulations, the update of the outer linked cell stripes is required. The same holds for the treatment close to process boundaries in distributed parallel MD simulations. For this purpose, the class \texttt{BoundaryTreatment} allows for traversing only the molecules in the boundary regions.

The single phases of the molecular dynamics algorithm, for example force evaluation, time stepping and boundary treatment, are put together in the class \texttt{MolecularDynamicsSimulation}. Modifications of the algorithm can be established by creating a new simulation class via inheritance from the existing molecular dynamics simulation and adapting the respective methods. This concept is used to create the \texttt{CoupledMolecularDynamicsSimulation} which represents a modified version of the simulation class for hybrid molecular–continuum simulations. Amongst others, it incorporates calls to modify mass or momentum inside the molecular dynamics simulation\footnote{The \texttt{MacroscopicCellService} of the macro-micro-coupling tool provides a method \texttt{distributeMacroscopicQuantities(int&)} See Sec. 13.4 for details.}.
The Peano framework arose within the scope of two PhD theses [133, 178] combining cache-efficient multi-level grid data structures with the functionality of a spatially adaptive Navier-Stokes solver. It is written in C++ and uses the space-filling Peano curve which allows for adaptive spacetree grid traversals for multiscale problems in arbitrary dimensions, see Fig. 9.1 on the left. The arising traversal scheme for cells and vertices of the underlying spacetree grid follows a modified depth-first traversal. It relies exclusively upon stack data structures and thus yields high data locality and high cache hit rates, respectively, independent from the specific application of interest. Besides the adaptive spacetree grid, a regular grid implementation is provided as well, allowing for the simulation on non-refined regular Cartesian grids. Before describing the Lattice Boltzmann application\footnote{Within the context of the Peano framework, an application is a separate component that uses the Peano kernel together with its own functionality and data structures.} within Peano (see Chap. 10) in detail, a nomenclature which characterises the different sub-components of each Peano application is introduced. It is mostly based on the concepts presented in [178]:

- **Peano vertices** and **cells** are the vertices and cells that define the nodes and spanned cells of the spacetree or regular grid. Each application may hold its own data structures for these vertices and cells. The data structures are typically modelled using the tool DaStGen [22] and its embedded scripting language. Each Peano vertex is strictly bound to its particular grid level of refinement. For example, if the grid consists of three levels as in Fig. 9.1, then there exist up to three Peano vertices at a particular position on the grid; in Fig. 9.1 on the right, this is for example the case for the upper left corner of the overall domain. There are three different types of vertices within Peano: **inner Peano vertices** denote vertices which are completely surrounded by inner Peano cells on the respective grid level, i.e., if $D$ denotes the spatial dimension, each inner Peano vertex has $2^D$ adjacent Peano cells that lie completely inside the computational domain. In contrast, **outer Peano vertices** are located outside the computational domain. They do not contribute to application-specific computations, but they are required for the grid traversal and setup. **Boundary Peano vertices** are adjacent to $2^D$ Peano cells from which a non-empty subset is located outside the computational domain. Besides this classification (inner, outer and boundary type), a second classification distinguishes between **persistent** and **hanging vertices**. **Persistent vertices** are completely surrounded by Peano cells and...
9. A SHORT INTRODUCTION TO PEANO

Figure 9.2: Structure of mapping for the regular (non-adaptive) Peano grid. The illustration shows the implementation of an exemplary mapping `RegularGrid2LogMapping` that is assumed to be connected to the `RegularGrid2LogAdapter`. The functionality of the mapping is restricted to printing log-statements: a log-statement is printed each time when a vertex is touched the very first and the very last time in a single grid traversal. Before the first and after the last vertex access, the vertex cannot exchange any more information with the adjacent cells or neighbouring vertices. Besides, log-statements for the cell-traversal are printed as well. After triggering the respective iteration over the grid from the runner of this application, the grid which is shown in the centre is traversed, and the implemented call-back methods from the mapping are automatically executed by the Peano kernel. The resulting output is shown on the right.

- One particular aspect of the Peano framework is the concept of mapping application-specific functionality onto the spacetree or regular grid. A mapping is a class that provides several methods that the application developer may implement and which are triggered during the grid traversal by the Peano kernel via a call-back mechanism. Each method provides local access to Peano cells or vertices. The corresponding grid traversal can be triggered from within the respective application. The exemplary structure of a mapping for the regular grid and the respective call-back structure is shown in Fig. 9.2. In order to reduce implementational efforts, the tool PeProt \(^3\) automatically generates mapping blue prints which are consistent with Peano’s predefined mapping signature.

- An adapter builds a collection of mappings that can be evaluated during a single grid traversal. An adapter may hold only one mapping (in this case, the nomenclature “mapping” and “adapter” becomes redundant and corresponds to the design presented in [178]) or several mappings whose functionalities are fully independent from each other. Adapters can also be generated via PeProt.

- For each application and corresponding grid structure, a repository of adapters exists

\(^3\)http://www5.in.tum.de/peano

hence have \(2^D\) neighbouring cells. In contrast, hanging vertices or hanging nodes have less than \(2^D\) adjacent Peano cells. This directly implies that a hanging node is always located on the finer grid level within the transition region between different grid levels, cf. Fig. 9.1 on the right.
that yields access to all functional units of the application. From within the application, a certain adapter can be chosen from the repository and applied to the grid data using the call-back principle from above. Hence, it is not the mapping that is directly triggered to be executed on all grid cells and vertices via the application, but rather an adapter that holds the particular mapping(s). The repositories are also automatically created using PeProt.

- **The state** of an application denotes another data structure modelled via DaStGen and contains global data that are required by the application. For example, a BGK-based Lattice Boltzmann solver needs a buffer for the relaxation time $\tau$ or a time step counter; these quantities are typically stored within the state object. With a BGK-based Lattice Boltzmann solver requiring exactly one relaxation time and one time step counter that are globally available, it can be further seen that each application instance exactly requires one state object. In case of parallel simulations, the state of a simulation is automatically distributed among the respective processes; its synchronisation (for example the synchronisation of the time step counter, referring to the BGK-Lattice Boltzmann solver) can again be accomplished via the implementation of a callback-method (mergeWithWorkerState(const State&)).

- From within a mapping, the only external object that can be directly accessed is the state object. “External” in this sense refers to an object that is not only available inside the mapping (such as a private member of a class), but an object that exists somewhere outside of the mapping’s frame and can be used to synchronise data of different mappings and algorithmic phases. In general, having the state for this purpose is sufficient from a software engineering point of view as the state may store all relevant pieces of information. However, similar to the Peano vertices and cells, the state in Peano is designed to hold (and synchronise) global data steering the overall algorithm of the underlying application. It is not designed to provide global functionality. The supply of the latter to the mappings, in turn, is often required and of elementary importance.

For example, consider flow simulations with dynamically changing geometries. It is trivial that a geometry that changes its form over time, e.g. due to fluid-structure interaction, needs to be synchronised and updated globally. Hence, it must not exist only inside one particular mapping. The geometry may be required by several mappings to allow for updating the flow field and the discrete fluid domain, correspondingly. Thus, a global access to the geometry is required, allowing for the access of the geometry from inside the mappings on the one hand and for the modification and synchronisation of the geometry object outside the mappings on the other hand. Such collaborations of mappings with external objects is realised within Peano via services. Services are singletons [61], i.e. they exist once per process, and provide access to global functionality from everywhere within a particular application.

- **The runner** class resembles the “main” of each application. Each runner creates its own grid and repository and yields access to all functional units that are required by the application. From the runner’s method run(), the method runAsMaster() is triggered which steers the execution of all algorithmic phases, i.e. the iteration of all adapters over the grid in the application-specific ordering. In distributed (MPI-based) parallel simulations, the method runAsMaster() is called on rank 0 only. All other ranks are scheduled according to a master-worker-concept: the main master on rank 0 distributes the work among the workers, and runAsWorker() is called on the respective processes. In order to allow for hierarchical work distributions, a worker can subsequently turn into a master for other processes and further distribute parts of its work. The synchronisation of the master and worker processes is established via hierarchical reduce- and broadcast-operations for the application-specific state object.
With these ingredients, a new Peano application is set up and used as follows (see also Fig. 9.3):

**Step 1: Data Structure and Application Modelling**

The Peano vertices and cells as well as the state of the simulation of interest are modelled in form of DaStGen [22] text files. Besides, a PeProt-compatible script is created. This script contains the locations of the vertex, cell and state declarations. Besides, it defines the names of the user-defined mappings and adapters and describes which mapping is merged.
into which adapter.

**Step 2: Creation of Application Template**
Executing PeProt using the script from Step 1 automatically generates a skeleton for the new application. The source code for the cell, vertex and state data structures, the repositories and the adapters is created. These classes are required by the Peano kernel and may not be modified by the application programmer. Besides, proxies for the Peano cells, Peano vertices and the state as well as templates for the application-specific mappings and the runners are generated. These classes build the interface structure and can be modified/implemented by the application developer according to the specific needs.

**Step 3: Implementation of Functionality and Algorithmic Phases**
Having the skeleton for the application at hand, the functional units of the application can now be implemented in separate classes. These classes can be instantiated from within the runner or the mappings of the application.

**Step 4: Integration of Functionality and Algorithmic Phases**
With the functional classes implemented, the single algorithmic phases of the application need to be put together. They are therefore embedded into the runner of the application.

**Step 5: Compiling, Executing and Postprocessing**
Finishing step 4, the code of the application can be compiled and executed. Afterwards, the data obtained from the application can be evaluated.

Following these five steps, an application is created within the Peano framework and filled with the required functionality.

Until now, the major focus with respect to the Peano kernel was put onto the supply of the spacetree and regular grid structure, including the respective traversal mechanism. However, it should further be mentioned that technical enhancements such as shared and distributed memory parallelisation are also encapsulated within the kernel. The parallelisation of an application can be achieved by implementing further call-back routines that are—similar to the grid traversal call-backs—automatically triggered by the kernel and so allow for the synchronisation of vertex and state data between the different processes. Depending on the particular needs of each application, special tuning strategies such as load-balancing techniques may be required to yield an efficient parallel version of the application-specific algorithm. In the case of the latter example, oracles can be implemented by each application independently, defining individual load-balancing strategies.

Further details about the concepts within the framework can be found in [178] and on the Peano webpages.

10 Lattice Boltzmann Implementations within Peano

10.1 Motivation

Having discussed the principles of the Peano framework, a (dynamically) adaptive Lattice Boltzmann implementation within Peano is presented in the following. The framework has been presented in previous works within the context of computational fluid dynamics [21, 133] where an adaptive finite element-based Navier-Stokes implementation has been studied and proven to be an efficient solver for different fluid dynamics scenarios on the macroscale. The analysis has shown that major parts of its efficiency are due to the high data locality and the local spatial adaptivity concept that are automatically provided by the Peano framework. Although the concept of local spatial adaptivity provides good means to reduce computational costs and increase computational accuracy, the simulations based on this Navier-Stokes solver may still be limited by the validity of the underlying continuum model, see amongst others the discussions and results in Sec. 5.7 and Chap. 15. Hence,
### Table 10.1: Functional requirements for multiscale flow simulations, listed and evaluated with regard to their realisation within the Peano framework.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Realisation in Peano</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spatial adaptivity</td>
<td>++</td>
<td>Local grid refinement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dynamic refinement and coarsening</td>
</tr>
<tr>
<td>Coupling to other solvers within Peano</td>
<td>+</td>
<td>Existing Navier-Stokes implementations within Peano</td>
</tr>
<tr>
<td>Coupling to other solvers outside Peano</td>
<td>+/-</td>
<td>Interfaces for structural solver coupling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Difficulties for “export” of spacetree grid</td>
</tr>
<tr>
<td>Two- and three-dimensional simulation support</td>
<td>++</td>
<td>Grid-kernel supports up to five dimensions</td>
</tr>
<tr>
<td>Modularity of applications and sub-components</td>
<td>++</td>
<td>Strict specification dictated by PeProt and Peano</td>
</tr>
<tr>
<td>Re-usability of code</td>
<td>+</td>
<td>Adaptations during code development may be necessary</td>
</tr>
<tr>
<td>Parallelisation</td>
<td>+</td>
<td>Common application parallelisation via call-back mechanisms</td>
</tr>
</tbody>
</table>

in order to bridge the gap between molecular and Navier-Stokes-based simulations, a mesoscopic Lattice Boltzmann solver may be highly desirable. Several requirements naturally arise for its implementation.

First, in contrast to numerous existing Lattice Boltzmann codes such as OpenLB\(^5\), Palabos\(^6\) or Exa PowerFLOW\(^7\) that are rather meant to build efficient alternatives to standard (Navier-Stokes-based) CFD-codes, the Lattice Boltzmann solver shall allow for the access of sub-continuum scales, including respective molecular effects such as Brownian motion (cf. Sec. 5.3). Second, a spatially adaptive implementation is desirable to retain efficiency in complex micro- and nanofluidic scenarios. Besides, the Lattice Boltzmann solver needs to be compatible to the micro- and the macroscale such that multiscale simulations become possible, ranging from the molecular to the continuum scale. Thus, a coupling to both molecular dynamics and Navier-Stokes solvers shall be possible. In case of the latter, technical issues may be strongly reduced, when both Lattice Boltzmann and Navier-Stokes implementations are based on the same implementation principles, i.e. the same grid layout and the same data structures.

With the Peano framework providing an environment for adaptive Cartesian grids and an existing efficient Navier-Stokes implementation, the idea emerged to develop an adaptive Lattice Boltzmann solver within this framework. Efficiency and compatibility aspects of the framework and Lattice Boltzmann automata are discussed in Sec. 10.2. The basic design and algorithmic incorporation of the Lattice Boltzmann application into Peano are presented in Sec. 10.3 and 10.4. The adaptive formulation and its realisation, including an extension to dynamic mesh refinement, is described in Sec. 10.5.

### 10.2 Efficiency and Compatibility

Peano provides several of the main pillars to high-performance multiscale simulations that have been previously mentioned in Sec. 2.3. The single requirements with particular regard to Peano’s functionalities are listed in Tab. 10.1 and are shortly discussed in the following.

One major feature of Peano is given by its spatially adaptive and multi-level representation of domains, using the spacetree grid implementation. Local grid refinement as well as dynamic refinement and coarsening that are triggered on-the-fly during a running simulation

\(^5\)http://www.openlb.org
\(^6\)http://www.palabos.org
\(^7\)http://www.exa.com
can be applied, mostly independent of the specific application of interest. This refinement
does not imply runtime overheads since the Peano curve-based traversal mechanism natu-
rally extends to newly created grid cells and vertices. Besides the corresponding new vertex
and cell objects, no additional memory is required to store or manage the dynamic adaptive
grid, retaining a low memory footprint.

In order to allow for the simulation on different scales or models, a coupling of different
solvers is required and needs to be integrated into a single piece of software. With existing
Navier-Stokes implementations at hand, a coarse-graining of a micro- or mesoscopic flow
description to the continuum scale can be established directly within Peano. The missing
links between the models such as the mapping of the unknowns from one to the other flow
description can be integrated into the respective solver setup and is straightforward; a cou-
pling strategy is to be discussed in Chap. 12 and has been published in [134]. However,
some adaptions in the code development may be necessary to reduce code duplications; see
Sec. 12.3 for details. Besides internal coupling mechanisms, a coupling to external codes
might be required. Respective interfaces can be implemented in form of services that are
independent from the particular Peano application and consequently can be re-used by dif-
ferent applications, cf. Chap. 9.

From a software engineering point of view, a modular software layout is highly desirable when
dealing with complex simulation scenarios where multiple solvers are involved. The same
holds for the design of a single solver if this solver is built upon several sub-components. As
an example, consider the Navier-Stokes solver described in [133]: with different space- and
time-discretisation schemes and various pressure Poisson equation solvers required, a modu-
lar and consistent way of integrating all of these sub-components into a common scheme
presents a major challenge. In Peano, a clear modularity concept is applied: each application
is encapsulated and separated from other applications and the kernel. Hence, there exist
several applications such as solvers for the Navier-Stokes equations or the heat equation that
are completely independent from each other. Within each application, the initial creation of
the application structure using PeProt (see Chap. 9) provides a code frame to the developer
which clearly separates the different aspects that are required by each application: data
structures, application-specific functionality, mapping this functionality onto the computa-
tional grid, etc. Thus, a high level of modularity is enforced for both application-internal
and inter-application design.

Simulations on two- and three-dimensional computational domains are possible within Peano.
In general, problems of arbitrary dimensionality may be tackled by the Peano kernel; test
cases for dimensions $D \in \{1, \ldots, 5\}$ are included in Peano's test suite. Besides, shared and
distributed memory parallelisation are provided via the Peano kernel. The required functionality—communication, synchronisation, etc.—is mostly hidden from
the specific applications and reduces to the implementation of additional callback-methods.
Following this discussion, it becomes apparent that Peano brings all key features that are nec-
essary for parallel multi-level (or multi-scale) simulations in arbitrary dimensions. Stepping
towards the particular application in form of the Lattice Boltzmann scheme, the individual
properties of the underlying scheme need to be re-visited and compared to what the frame-
work offers.

From the Lattice Boltzmann theory discussed in Chap. 5, it can be seen that standard
Lattice Boltzmann schemes work on Cartesian grids with quadratic (2D) or cubic (3D)
cells; this generally fits to the Peano grid implementation. However, in contrast to standard
Cartesian grids, the Peano grid does not allow direct indexing of neighbouring cells or
vertices. Instead, the traversal scheme allows for direct neighbour communications and thus
accomplishes a particular access pattern to local and neighbouring data. This layout stands
in contrast to most standard LB codes where regular Cartesian grids are used and traversed

---

8The coupling mechanisms described in [134] are based on the original Peano implementation from [178].
Most of the underlying ideas can be naturally extended to the current Peano version that has been discussed
in Chap. 9.

9See http://www5.in.tum.de/jeimz/piano for details on the respective unit and integration tests of the
framework.

9It shall be noted that Lattice Boltzmann approaches for irregular lattices have also been proposed in
literature, see amongst others [97, 132].
Figure 10.1: Block structure of the Lattice Boltzmann solvers in Peano [137]. The dashed line represents one Peano cell, the filled black circles denote the respective Peano vertices. In this example, each Peano vertex holds $6 \times 6$ Lattice Boltzmann cells.

generating to lexicographic ordering of the data [179]. Further problems arise from the stack-based traversal scheme that is used on the Peano spacetree grid. First, the stack-based approach requires the copying of the vertex and cell data between the different stacks that are used by the Peano kernel. For purely computation-bound problems, i.e. problems that are completely dominated by (mostly) local computations, this property does not imply any restrictions. In the case of Lattice Boltzmann implementations that require a huge amount of memory and memory transfer operations as well as a moderate number of computations per degree of freedom, these copy operations significantly contribute to the overall computing time and may even become the dominant factor; see [125] and the discussion on performance of different Peano-based Lattice Boltzmann implementations in Chap. 14. Second, as the traversal of the Peano grid is based on the iterates of the space-filling Peano curve, the determination of the curve is required after leaving and before entering a particular grid cell. This yields additional integer operations that are required to be executed for each Peano grid cell in the spacetree and thus contribute significantly to the overall computational amount of work that is required by a Lattice Boltzmann simulation. The related computational efforts may be reduced for static adaptive grids by pre-computing the direction of the curve in a grid initialisation phase; during the simulation, the direction could then be determined for each cell via a lookup-table. However, for three-dimensional scenarios, the size of the respective lookup-table may become very huge\footnote{Arturo Mora, KAUST Student exchange project}. Besides, a pre-computation is not possible anymore in case of dynamically changing grids. One more drawback of the traversal scheme consists in the fact that it is always the whole spacetree that needs to be traversed. Single cells or vertices or subgroups of cells and vertices (such as vertices and cells of only one tree level) cannot be traversed. For spatially adaptive Lattice Boltzmann methods, this implies that for each time step on the finest grid level, the whole tree needs to be traversed, though accessing only the finest grid level would be sufficient.

Hence, several aspects that are required for efficient Lattice Boltzmann implementations are not natively provided by Peano. Still, the multi-level and multi-dimensional nature of Peano, together with its modularity and existing solvers incorporated, make the framework very attractive for multiscale considerations. Concluding this discussion, solution strategies to reduce the efficiency bottlenecks for Lattice Boltzmann simulations within Peano are developed in the following section.

10.3 The Block-Structured Lattice Boltzmann Application in Peano

With Lattice Boltzmann methods gaining best performances on regular blocks and as most adaptive LB solvers work on whole blocks of data, a block storage scheme can be applied in a similar fashion within the Peano framework. The respective structure is illustrated in Fig. 10.1. Each Peano vertex holds a block of $N^D$ Lattice Boltzmann cells. The Lattice Boltzmann cells can be processed similar to native Lattice Boltzmann simulations. The data are stored within each block following the collision-optimised memory layout [179]. For each
Lattice Boltzmann cell \(\mathbf{x}\), the distributions \(f_i\) are stored contiguously in memory; the cells are sorted lexicographically. This layout is known to show performance drops for bigger block sizes due to excessive cache misses in the streaming operations. The major parts of the following simulations, however, operate on smaller blocks. Besides the distributions, additional memory is used to store the local velocities \(\mathbf{u}\) and density values \(\rho\) which facilitates post-processing of these data as well as the density interpolations required for the particle-fluid interactions (see Sec. 5.6).

Streaming or non-local operations that span over the boundaries of one block require interactions with neighbouring blocks, that is neighbouring Peano vertices. These interactions can be realised via the `enterCell(PeanoCell &cell, PeanoVertex vertices[TWO_POWER_D])` callback-methods of the Peano mappings. Alternatively to the presented approach, the blocks could also be stored within the Peano cells. The communication between neighbouring blocks in this case, however, becomes more complex\(^{12}\). Therefore, a block structure with respect to the Peano cells is not further discussed.

Besides having regular blocks to efficiently execute the Lattice Boltzmann operations on, the block structure also has the advantage that the relative costs for the tree-traversal along the space-filling curve become significantly smaller. If a total domain resolution of \(M^D\) Lattice Boltzmann cells is required, the number of Peano cells on the finest tree level is of the order \((M/N)^D\). As a consequence, the number (and the respective cost) of the overall space-tree-traversal computations is reduced by a factor of \(N^D\) on this tree level. The same arguing holds for all grid levels, so that \(N^D\) is the total reduction factor for the traversal costs of the Peano kernel. However, local grid refinement may now only be applied in a blockwise manner and cannot be defined via the Peano framework for each Lattice Boltzmann cell. This generally does not pose severe restrictions, since the block size may be adapted depending on the scenario of interest. The block size hence represents a tunable parameter that can be adjusted. This yields a compromise between efficiency and local refinement properties.

Another problem described in the previous section is the huge number of copy operations of Peano vertex and cell data between the different stacks of the Peano kernel. As mentioned in Chap. 9, the stack data structures allow for high cache efficiency in the underlying simulation. For small vertex and cell data, the copy operations are strictly executed within the cache of the underlying hardware system. Hence, they are very cheap. The Lattice Boltzmann blocks, however, may easily exceed the cache size. For example, assume a three-dimensional Lattice Boltzmann simulation based on the D3Q19 velocity discretisation, using a block size \(N = 6\). Storing one set of the distributions, the cellwise fluid density and velocity results in \(6^3 \cdot (19 + 1 + 3) \cdot 8B \approx 39\) kB. Considering the Intel architecture i7-870 which is also to build the platform for the performance evaluations in Chap. 14, the L1 cache has a size of 32kB and the L2 cache a size of 256kB per core. One single block hence already exceeds the L1 cache. Neighbouring block communications which in the Peano context involve \(2^D\) blocks exceed the L2 cache. Concluding, the large size of the blocks together with the stack-based copy operations induce cache misses, contradicting the basic idea behind the (cache-efficient) stack principle and data locality claimed by Peano. For these reasons, storing the Lattice Boltzmann blocks directly within the Peano vertex records does not represent a good strategy and has to be avoided. With the Peano grid representing rather a stack data structures; instead, only a block identifier is stored within the Peano vertex record. This identifier is kept consistent throughout the simulation, due to the Peano paradigms for its cell and vertex data. Hence, the block of data that is associated to this identifier and is stored within the Lattice Boltzmann application can also be kept consistent, as long as the access to the data is only provided via the mappings, i.e. as long as the data are traversed along the iterates of the Peano curve. This also dictates the only drawback that cannot be resolved by the presented scheme: to retain consistency, a traversal of sub-parts of the tree may not be allowed. In order to provide access and manage the single Lattice Boltzmann blocks, a `GridManagementService` is introduced in the Lattice Boltzmann application.

\(^{12}\)Within Peano, one has no access to more than one Peano cell on a particular grid level at a time during the tree traversal. This is in contrast to the `enterCell(\ldots)` method that is used for interactions of neighbouring vertices.
Figure 10.2: Two grid levels of a locally refined grid for block sizes $N = 12$ and $N = 5$. The filled orange circles denote the Peano vertices on the fine grid, the orange lines represent the corresponding Lattice Boltzmann cells. As twelve can be divided by three, all fine grid Lattice Boltzmann cells that are contained within one coarse grid Lattice Boltzmann cell belong to one particular fine grid Peano vertex. For $N = 5$, this is not the case.

capable of allocating and deleting blocks from memory and implements the block identifier-to-block data index mapping.

Finishing this section, some remarks follow on the choice of valid block sizes $N$. Due to consistency in the spatial grid refinement, $N$ is required to be a multiple of three. The reason for this is explained in detail in the subsection 10.5.1 and is shortly depicted in Fig. 10.2: If $N$ cannot be divided by three, the fine grid Lattice Boltzmann cells that are located inside one coarse grid Lattice Boltzmann cell may not belong to only one fine grid vertex. They are logically related to several fine grid vertices, complicating the refinement process.

10.4 Implementation of the Lattice Boltzmann Algorithm

In the following, the implementational concepts of the Lattice Boltzmann algorithm are described. An overview of the Lattice Boltzmann application is given in Fig. 10.3: two- and three-dimensional scenarios with different velocity discretisations are supported on both regular and adaptive grids. Currently, the D2Q9, D3Q15, D3Q19 and D3Q27 schemes are available and can be chosen at compile time. This allows for fast prototyping of new models that might be developed and described only for one particular velocity discretisation in literature; although a conversion to other velocity discretisation schemes in most cases is possible and often straightforward, the derivation of the underlying sets of equations and expression can be a time-consuming task. Besides, depending on the memory that is available on particular machines and the memory and simulation time requirements of the simulation scenario of interest, the one or the other velocity discretisation scheme might be preferable. For example, a three-dimensional simulation that requires a huge number of time steps and for which the simple BGK collision model and the D3Q15 discretisation are sufficient is solved much faster via the D3Q15 discretisation than applying the D3Q27 model. Different collision models and boundary conditions can be applied, depending on the scenario and scale of interest. The respective implementations, including a particular description of the streaming step implementation, are presented within this section; see Sec. 10.5 for implementational details on the spatially adaptive Lattice Boltzmann scheme.
Figure 10.3: Overview of the Lattice Boltzmann application within the Peano framework [137]. Simulations in two and three dimensions are supported for different velocity discretisation schemes. The user can choose between the usage of regular, that is non-adaptive, grids, and locally adaptive grids. In case of the latter, an extension for dynamic grid refinement allows for the simulation of moving geometries. Besides, different collision kernels and boundary conditions can be applied, depending on the requirements of the specific flow problem to be solved.

10.4.1 Collision Models

As mentioned in Sec. 5.3, the collide step only involves the distribution functions $f_i$ within one particular Lattice Boltzmann cell at position $\mathbf{x}$. A clear interface for the collision process is simple to be realised: a class `AbstractCollisionModel` provides the abstract method `void collide(int cellIndex,...)` which is meant to carry out the local collision process within one particular cell at `cellIndex` inside a Lattice Boltzmann block. A new collision model simply inherits from `AbstractCollisionModel` and implements this method. Hence, the incorporation of the collision operators for the BGK, MRT and FLB models is straightforward. However, the developer is responsible for the support of the different velocity discretisation schemes. For the collision models from Sec. 5.3, the following combinations of collision models with velocity discretisations are allowed:

- **BGK model**: D2Q9, D3Q15, D3Q19, D3Q27
- **MRT model**: D2Q9, D3Q19
- **FLB model**: D2Q9, D3Q19

The collide step shall only be carried out for inner cells. For outer cells and for fine grid overlap cells in the adaptive case, the collide step shall be skipped. This is established via the `CollisionModelWrapper` which holds pointers to different collision model implementations. Depending on the inner-flag and the overlap information of a cell, the correct collision model pointer is addressed via a lookup-procedure. For outer and fine grid overlap cells, the respective collision model implementation corresponds to an empty method `void collide(int cellIndex,...){}`. The `CollisionModelWrapper` further encapsulates the traversal of the whole cell block.

Depending on the domain resolution in adaptive flow simulations, different models for physical effects can be used. For example, Brownian fluctuations significantly reduce between subsequent grid levels. With the coarser cell having a volume of $3^2$ fine cells and being...
updated over a time interval which is three times larger than the fine grid time step, the fluctuations are strongly damped. Dünweg et al. showed that the scaling of the fluctuations that are applied in moment space is proportional to $\sqrt{dt^2/dx^{D+2}}$ [42]. This proportionality is a little more stringent than the viscosity scaling between different grid levels (which is given by $dt/dx^2$). As a consequence, the fluctuations rapidly decrease when going to coarser grid levels and may be cut off on the coarsest grid levels of a simulation. In this case, a modification or replacement of the MRT-based fluctuating model can be introduced, yielding a further reduction of the computational costs; details on this example are described in Chap. 16. Hence, different collision model implementations shall be applicable on different grid levels. The `BlockCollisionModelManager` provides the respective functionality: via the configuration file of a simulation scenario, the user can define the range of the applied collision models, that is the maximum and the minimum Lattice Boltzmann mesh size for which a particular collision model shall be applied. The `BlockCollisionModelManager` holds one object of the `CollisionModelWrapper` for each simulation grid level. It hence provides access to the correct model on each level. The `BlockCollisionModelManager` is a member of the `RegularGrid2RegularBlockSolver-` and the `SpacetreeGrid2RegularBlockSolver`-mappings, steering the basic and adaptive Lattice Boltzmann algorithm. The `collide` step can thus be invoked from the `touchVertexFirstTime(...)` callback of these mappings.

### 10.4.2 Boundary Treatment

Similar to the collision model implementation, the implementation of boundary conditions is established. The interface `AbstractBlockBoundaryCondition` allows for the implementation of methods `treatBoundaryBeforeCollision(...) ` and `treatBoundaryAfterCollision(...) ` which are triggered before and after the collision procedure, respectively. Both methods obtain the density, velocity and the distribution block data as arguments. Besides, a `BoundaryData` object and the index of the distribution function that needs to be constructed at this boundary are handed over to these methods. The `BoundaryData` object holds additional boundary information like the velocity at the channel inlet or the fluid density value at a pressure outlet boundary. In order to provide different types of boundary conditions, a `BlockBoundaryConditionManager` object within the solver mappings holds objects of all boundary conditions. Access to the conditions is provided in the `touchVertexFirstTime(...)` callback method.

For simple boundary schemes such as the ones presented in Sec. 5.5, the description from above is sufficient. In case of more complex formulations of boundary conditions, however, further data access may be required; this particularly holds for inter- and extrapolation-based schemes where a local boundary treatment is not possible anymore. The extrapolation scheme at pressure outlets which is defined within the context of finite Knudsen number simulations in Sec. 5.7 is one example for the latter challenge. However, as this scheme is only used for testing purposes in simple channel-like setups, a local treatment within a particular Lattice Boltzmann block is possible for these scenarios; a synchronisation with neighbouring blocks is not required. For more general scenarios, the local pressure outlet condition described in Sec. 5.5 can be used.

### 10.4.3 Streaming

Different methods have been proposed over the years to implement the streaming step in Lattice Boltzmann simulations [8, 109, 118, 124, 125, 134, 146]. For a comparison of the different schemes with respect to performance considerations, see amongst others [181]. With the special callback structures of Peano dictating the access pattern of neighbouring fluid cells, a new streaming pattern has been developed within the scope of this thesis for the Lattice Boltzmann application [125, 134]19. The pattern originally was developed

---

19 At about the same time, an analogous method was developed independently in the group of Prof. Krajczyk at the Technische Universität Braunschweig in the context of Lattice Boltzmann implementations for GPUs [118].
assuming the identity of Peano and Lattice Boltzmann cells and later was extended to a blockwise vertex-based evaluation of the Lattice Boltzmann scheme. In order to explain the methodology, assume the Lattice Boltzmann blocks on the Peano vertices to have a size of $N = 1$, i.e. each block and thus each Peano vertex exactly holds one Lattice Boltzmann cell. The streaming process is illustrated in Fig. 10.4: the collision process is finished within the $\text{touchVertexFirstTime}(\ldots)$ method and therefore builds the starting point of the streaming. The $\text{enterCell}(\ldots)$ callback method is used to swap the distribution densities between the Peano vertices that are adjacent to the respective Peano cell. After traversing all Peano cells that are adjacent to a respective Peano vertex, the distributions have arrived at the Peano vertex, but are located on the wrong position in memory; the distribution $f_i$ is stored on the position of $f_j$. Therefore, the method $\text{touchVertexLastTime}(\ldots)$ is used to swap these distributions locally.

The typical improvements arising from this scheme comprise several aspects. First, only one set of distributions needs to be stored, compared to the simple, but most common, two-grid implementations. This significantly reduces the memory footprint of the Lattice Boltzmann application which in this sense also fits to the Peano philosophy. Second, stepping from the Peano implementation to other Lattice Boltzmann codes, a (shared memory) parallelisation of this scheme is for free. The loop over the elements that are spanned by the Lattice Boltzmann cell’s midpoints can be parallelised straightforward, making the scheme also attractive for multi-core architectures, cf. [118].

With the Lattice Boltzmann application storing blocks of LB cells on the Peano vertices, an extension of the streaming step mechanism from above to the block structures is necessary. There are different ways to accomplish this extension. One possibility, which is currently used in the code, is to perform all streaming operations that can be carried out locally within a block after the collision step inside the $\text{touchVertexFirstTime}(\ldots)$ method. All other swapping operations are performed according to the streaming pattern from Fig. 10.4 from the $\text{enterCell}(\ldots)$ function calls, cf. Fig 10.5. The LB cell-local swappings are finally triggered as described above from the $\text{touchVertexLastTime}(\ldots)$ method.

10.5 Implementations on the Spacetree Grid

Having described the implementations of the standard steps in the LB algorithm, the considerations are now extended to the adaptive case using Peano’s spacetree grid. The interpolation-based and volumetric adaptive LB schemes have been explained in Sec. 5.4. The adaptive implementations within Peano are based on the volumetric approach. Currently, the first-order scheme is implemented which—as illustrated in Fig. 5.3—uses strictly local data evaluations in the prolongation and restriction steps between different grid levels; an extension to the second-order scheme can be established similar to the following descriptions.

The reasons to use the strictly local first-order schemes arise from both computational and physical considerations. First, from the computational point of view, abstaining from interpolations yields higher data locality and thus allows for higher performance in the com-
Figure 10.5: Streaming step using the callback mechanisms in Peano and a block size $N = 6$. The local streaming inside each block is carried out during the callback $\text{touchVertexFirstTime}(\ldots)$. When entering a (green-coloured) Peano cell, the green-coloured distributions are exchanged between the respective $2^D$ Peano vertex blocks. Finally, all distributions have arrived in the correct cell and need to be locally swapped to their correct location during $\text{touchVertexLastTime}(\ldots)$. 

75
putations. Compared to the interpolation-based methods, there is also no need for temporal interpolations which further reduces the required number of distribution sets that need to be stored. Thus, the memory requirements are significantly smaller in this case. Second, as mentioned throughout the previous discussions on nanoscale flow modelling, the incorporation of Brownian fluctuations may be of essential importance for nano-sized scenarios. These fluctuations are typically modelled via Gaussian noise and are assumed to be uncorrelated in both space and time. The interpolation-based schemes would automatically introduce strong correlations, arising from the interpolation of the fine grid values from several coarse grid cells. The volumetric first-order approach also cannot strictly avoid local correlations on the fine grid as the post-collision populations of the coarse grid are homogeneously distributed inside the fine grid overlap region. However, these correlations are induced via a single coarse grid cell only; hence, no additional correlations can occur from the interplay of neighbouring coarse grid cells as it is the case in the interpolation-based approach. Besides, the exact conservation of mass and momentum is considered to be a major feature of the volumetric scheme which may particularly facilitate the control over these macroscopic quantities in case of coupled simulations.

In the following, implementational details on the volumetric scheme are described. The implementation of the volumetric approach on the spacetree grid is explained in the subsection 10.5.1. An extension of this scheme to dynamically changing grids is discussed in the subsection 10.5.2.

10.5.1 Static Adaptivity

Going to adaptive grids, the standard steps in the LB algorithm—collision, streaming and boundary treatment—need to be extended by prolongation and restriction steps, allowing for the transport of the distribution functions between different grid levels. Re-considering the volumetric algorithm from Sec. 5.4, several requirements and challenges to the implementation can be formulated:

1. An overlap layer needs to be introduced and incorporated into the Peano tree structure.

2. In order to keep the data access local between coarse and fine grid cells, the prolongation of data from the coarse to the fine grid needs to be performed before the streaming step is executed on the coarse grid. Otherwise, distributions may have left the coarse grid cell and can only be accessed via the respective neighbouring coarse grid cell. Similarly, the restriction of data needs to be established after the streaming step is executed to avoid overwriting the post-collision data in the coarse grid overlap cells.

3. Due to the nested time stepping on the finer grid levels, one can estimate the minimum number of grid traversals for one time step on the coarse grid. Assume a simulation involving \( L \) simulation grid levels. With a refinement factor \( r = 3 \) for the Peano grid, at least \( 3^L \) traversals are required to accomplish the collide-stream algorithm on the finest grid level. Hence, a challenge is the minimisation of grid traversals and respective data accesses for the adaptive LB scheme.

4. The “jump” between different grid levels \( l, k \) should not exceed one, in order to fit the volumetric scheme. This means that the mesh size ratio of two neighbouring grid cells fulfills \( dx^{cell_1}/dx^{cell_2} \leq 3 \), assuming an ordering of these mesh sizes \( dx^{cell_1} \geq dx^{cell_2} \).

The incorporation of the overlap layer is to be addressed first. Within the overlap region, the distributions from the coarse grid need to be prolonged to the fine grid and subsequently streamed into the fine grid region. Besides, the distributions stemming from the fine grid are accumulated within the overlap region, averaged and written back to the coarse grid. The natural choice where to place the overlap layer would be the LB blocks defined at the hanging Peano nodes. The overlap could be easily detected as hanging nodes are separated from the other Peano vertices with respect to their callback methods. Each time that the mapping-specific callback functions `createHangingNode(...)` or `destroyHangingNode(...)` are triggered, the prolongation or restriction of the data on this node could be carried out.
With the hanging nodes getting destroyed after each grid iteration, it would be required to immediately write the distributions to the coarse grid after each streaming step on the fine grid. The requirement no. 2, however, yields that an immediate restriction of the distributions and storage at their final location within a coarse grid cell is not possible yet. Otherwise, the distributions from the coarse grid overlap would be overwritten since—according to requirement no. 2—they cannot be streamed immediately. Therefore, this default methodology does not work for the adaptive Lattice Boltzmann application when using a single set for the distributions\(^\text{14}\).

Hence, a different approach is used to detect the overlap region and describe the mapping of distributions within this region. Instead of placing the overlap region on the hanging Peano nodes, it is placed on the first layer of consistent fine grid Peano vertices. With the overlap located on consistent non-hanging vertices, the data inside the overlap can be kept consistent over several fine grid time steps. An immediate mapping of fine and coarse grid distributions in each fine grid time step is consequently not required. In contrast to placing the overlap region on the hanging nodes, it is a priori not clear which Peano vertex on the fine grid belongs to the overlap region. A flagging of the respective fine grid overlap cells can be achieved via the integration of the hanging nodes into the streaming process. All distributions of a hanging node are initialised by the value -1. With (-1)-entries representing unphysical values for the LB distributions\(^\text{15}\) that cannot occur within the context of LB simulations, the following statement can be formulated:

Let hanging node blocks also participate in the streaming process. If the streaming operation is performed \(\leq 3\) times on the fine grid of an adaptive LB simulation within Peano, each LB cell that afterwards holds a (-1)-entry is either located on a hanging node or is part of the overlap region.

Besides, depending on the index \(i\) of the respective distribution \(f_i = -1\), it can further be determined in which direction the (non-overlap) fine and coarse grid cells are located. Therefore, consider the example from Fig. 10.6: at the upper boundary of the only non-hanging vertex, the distributions belonging to the lattice velocities \((-1, -1)\), \((0, -1)\) and \((1, -1)\) contain (-1)-entries and consequently are part of the overlap region. The same arguing holds for the right boundary stripe of the non-hanging vertex. Here, the lattice velocities \((-1, -1)\), \((-1, 0)\) and \((-1, 1)\) transport the (-1)-entries into the overlap domain.

A further remark shall be made on the technical realisation of this approach in terms of additional memory requirements. From a naive point of view, the flagging procedure from above would require a whole block layer surrounding the fine grid domain where each block is filled with (-1)-entries. As one is not interested in the data that are streamed into the hanging node blocks\(^\text{16}\), it is sufficient to reserve \(2^D - 1\) particular hanging node buffer blocks inside the GridManagementService. These buffers are allocated globally once and are reused in each non-local streaming operation where hanging nodes are involved: each time when a hanging node is touched in the traversal process the service resets the values for one of the buffers and returns a temporary reference to its data.

The approach from above yields a straightforward flagging of the overlap region. This particularly holds for dynamically changing grids, where hanging nodes are created or deleted. The streaming of (-1)-entries thus changes and can be tracked on-the-fly; for details on this case, see the explanations in the subsection 10.5.2. The tracking of the overlap layer, however, comes at a price: each Lattice Boltzmann cell needs to be checked for (-1)-entries in order to determine whether restriction or prolongation techniques need to applied and which particle distributions are affected. In the case of static grids, this checking may be reduced by only considering Peano vertices that are located close to hanging nodes. However, a further reduction in computational costs, for example in terms of an a priori evaluation

---

\(^{14}\)Note that this algorithm works again in case of using two distribution sets per LB cell. Then, the distributions from the fine grid can be written into the "streaming" set of the coarse grid cells. The post-collision distributions from the coarse grid belong to the "collision" set which remains untouched during the restriction process.

\(^{15}\)Distributions are required to be strictly non-negative.

\(^{16}\)The physical collide-stream algorithm in these regions is carried out on the coarse grid.
and storage of the overlap relations may be desirable. This can be established easily in one preparatory iteration over the newly created static adaptive Peano grid. Additional storage is required for each Peano vertex which—particularly for simple grid setups—does not significantly contribute to the overall memory requirements that are mainly dictated by the size of the particle distribution fields. This solver strategy has also been implemented and optimised to solve problems on static grids. From the construction of the overlap region, the fulfillment of requirement 2 is straightforward: as the data in the overlap region is kept consistent over several grid iterations, the prolongation of the data can be carried out during the first fine grid time step, triggered for example from the touchVertexFirstTime(vertexFromFineGrid, ...) callback. Afterwards, the coarse grid data is not read anymore from the fine LB cells. Hence, the streaming step on the coarse grid can for example be performed during the very next fine grid time step as illustrated in Fig. 10.7. The restriction of the fine grid data within the overlap region can be achieved after the last fine grid time step is finished. In the current implementation, the restriction is performed from the touchVertexLastTime(..., ...) callback of the respective solver mapping after the local swapping of the distributions is finished. It can hence be carried out in the same grid traversal.

From Fig. 10.7, it can be observed that for the two-level grid, the solving of the adaptive LBM algorithm can be achieved with the minimal number of Peano grid traversals, i.e. the number of grid traversals equals the number of fine grid time steps in this case. The re-
Figure 10.7: Adaptive LBM algorithm within Peano, illustrated for a one-dimensional two-level grid. The light-grey region marks parts of the hanging node domain, the blue-coloured domain represents the overlap region. The streaming on the coarse grid is shifted by one iteration and is executed during the second fine grid time step to avoid read-conflicts during the prolongation of the overlap data. In total, three Peano grid traversals are enough to implement the whole adaptive algorithm in this example.

The respective methodology can be extended to grids with arbitrary numbers of grid levels \( L \); the number of grid traversals in all cases is given by the number of fine grid time steps \( 3^L \). In this sense, the number of grid traversals is kept minimal.

The only point that has not been addressed so far is requirement 4, requiring a continuous embedding of subsequent grid levels into each other. With the refinement triggered due to geometry queries throughout this work, the solution strategy to obtain a respective grid structure is achieved via the default criterion of geometry-based refinement within Peano. The respective refinement method is given in Alg. 10.1. Here, the method isComplete-

\[
\text{define position } x \text{ of Peano block, mesh size } h \text{ of Peano grid, current grid level } l
\]

\[
\text{function } \text{bool refine}(x, h, l)
\]

\[
\text{return } !\text{geometry.isCompletelyInside}(x, h)
\]

\[
\&\& !\text{geometry.isCompletelyOutside}(x, h)
\]

\[
\&\& (l < \text{MAXIMUM_NUMBER_OF_GRID_LEVELS})
\]

\[
\text{end function}
\]

\[
\text{lyInside}(x, h) \text{ returns true, if no geometric boundary is detected within a box of size } 2h
\]

which is put around the position \( x \). Analogously, the method isCompletelyOutside\((x, h)\) is defined. Hence, if the block is not located completely inside or completely outside of the computational domain, a refinement is triggered. In order to suppress an infinite number of refinement steps at the boundaries, a (predefined) maximum number of grid levels is set a priori which stops the refinement at a certain grid level. A similar formalism can be used to allow for other refinement criteria such as maximum gradient or extremal value detections.

10.5.2 Dynamic Grid Refinement

Although the descriptions from the previous subsection allow for LB simulations on spatially adaptive grids, they are currently designed for static grids only. Hence, once the grid is generated during the initialisation phase of the LB solver, it cannot be changed or adapted to the scenario. Especially for highly unsteady scenarios whose characteristics strongly vary over space and time, a dynamic grid adaption—also denoted as adaptive mesh refinement (AMR)\(^{17}\)—is highly desirable. Using AMR, high resolutions can be dynamically in-

\(^{17}\)Although adaptive mesh refinement may comprise dynamic or static adaptivity, it is used in the context of dynamic mesh refinement in the following.
Refinement Coarsening

Figure 10.8: Dynamic refinement and coarsening in the volumetric LB algorithm [137].

Introduced where necessary, and coarser grid resolutions may be applied in the other regions. However, only little effort has been spent so far on the development of dynamically adaptive Lattice Boltzmann schemes. Wu and Shu developed a stencil adaptive LB algorithm and provide results for two-dimensional cavity and channel flows [183]. Different single- and multiphase simulation results, using AMR techniques for the interpolation-based adaptive LB algorithms, are provided in [89]. A further approach applied to multiphase problems is discussed in [172]. Yu provides first results in the context of volumetric multiphase simulations on changing grids in [188]. Starting from the volumetric scheme described in the previous subsection and the respective overlap layer definition, a dynamic variant of this method is developed in the following which particularly fits to the local cell and vertex concept of the Peano framework. The idea is sketched in Fig. 10.8. Dynamic refinement shall be discussed first: if a region (red-coloured cell in Fig. 10.8 on the left) shall be refined in the next (coarse grid) time step, new Peano vertices are introduced. The creation of the new vertices is straightforward, due to Peano’s local refinement features. In order to fill all LB cells near the boundary region with valid data, both the original and the newly created overlap layer are filled with the post-collision states from the coarse-grid level. Afterwards, the adaptive algorithm is executed analogously to the static case. Hence, the original LB overlap cells on the fine grid are now considered and handled as regular fine grid cells, and the newly created LB cells provide the new overlap area.

In a similar fashion, the coarsening can be applied dynamically, cf. Fig. 10.8 on the right: before coarsening the (red-coloured) LB cells and respective Peano vertices on the fine grid, both the distributions within the overlap area as well as the distributions in the neighbouring fine grid region are averaged and written back to the LB cells on the coarser level. Removing the original fine grid overlap cells results in a fully determined set of distributions in both the red-coloured coarse grid cell (green and black distribution) and the coarse grid cell in the newly defined overlap layer (grey and blue distribution).

The static adaptive LB algorithm described before has been extended by this AMR technique. Therefore, different states for the Peano vertices—and hence for the LB blocks—are introduced. The states are called Default, IsCoarsenedVertex, IsNewPersistentVertex and HasNewPersistentSubVertices. Vertices that do not obtain a specific flag out from the latter three states during the dynamic refinement process have the Default state. A new Peano mapping SpacetreeGrid2DynamicRefinement is called after each fine grid time step of a certain coarsest grid time step interval. However, the dynamic refinement is not required to be used in all coarsest grid time step intervals; a measure for the respective refinement intervals is discussed later within this subsection.
Figure 10.9: Automatic refinement by the Peano kernel. The black dots describe Peano vertices that are refined since they are located close to the static grey-coloured geometry. The green dots represent Peano vertices which are refined due to their proximity to the green-coloured moving sphere. With the sphere moving downwards, the red-coloured vertex can be coarsened as its bounding box of $2h$ is completely inside the fluid domain. However, all surrounding vertices are refined. Since isolated hanging nodes are not allowed in Peano, the red vertex is automatically refined by the internal routines of the Peano-kernel although the application may trigger a coarsening at this position.

The mapping $\text{SpacetreeGrid2DynamicRefinement}$ determines for each Peano vertex, if a refinement or coarsening is required. If a refinement is triggered on a vertex, the refinement is carried out immediately; we assume at this stage that this vertex is defined on grid level $l$. The refined vertex is flagged as $\text{HasNewPersistentSubVertices}$, since it typically embeds new fine grid LB regions. The Peano vertices that are created on the finer grid level during this refinement step are flagged as $\text{IsNewPersistentVertex}$. Until the next prolongation is performed between the grid levels $l$ and $l+1$, the original grid structure is used. The $\text{IsNewPersistentVertex}$-flagged vertices are therefore initialised by (-1)-entries and contribute to the simulation in the same way as hanging nodes. In the next prolongation between these levels, the information from the coarse grid is transferred to the finer grid. At this, all distributions within the fine grid region are constructed from the post-collision states on the coarser grid level. The distributions within the original overlap layer that are required from the coarse grid are missing at this stage of the algorithm as well; they are also constructed from the coarse grid post-collision distributions, respectively. Note that the distributions that were streamed from the original fine grid region into the original overlap layer of the fine grid are already available. Hence, a corresponding construction of these distributions from the post-collision distributions of the coarse grid is not required. When the prolongation is finished between the respective fine and coarse grid vertices, the flags $\text{HasNewPersistentSubVertices}$ and $\text{IsNewPersistentVertex}$ are removed, and the vertices are set to Default state.

If a coarsening is triggered on a Peano vertex on level $l$ to remove the level $l+1$, the vertex is flagged as $\text{IsCoarsenedVertex}$. The original grid structure needs to be used until the next restriction can be carried out within the original overlap; otherwise, the set of data to construct the missing coarse grid distributions is not complete yet. Hence, the coarsening and related deletion of the respective sub-vertices on level $l+1$ is not triggered immediately, but shifted to the fine grid time step where the next restriction is performed. Besides, the coarsening is only allowed if the Peano vertex on level $l$ has none of the Peano-internal refinement flags set. Assume we trigger a coarsening on a refined Peano vertex which is completely surrounded by refined vertices, see for example the red-coloured vertex in Fig. 10.9. The fine grid Peano vertex which is located at the same position as the coarsened coarse grid Peano vertex would—from the application’s logics—be converted into a hanging node. However, the Peano kernel automatically detects these situations and does not allow isolated hanging nodes. It corrects the respective Peano-internal grid flag of the coarse grid.
vertex and automatically triggers a refinement of this vertex. Thus, the coarse grid vertex is refined again, although the Lattice Boltzmann application triggered a coarsening. After the next restriction between levels \( l \) and \( l + 1 \) is carried out, the coarsening is performed and the fine grid vertices are deleted.

From these technical descriptions, it can be seen that the incorporation of the dynamic refinement technique yields additional grid traversals, due to the creation and removal of Peano vertices and cells. Depending on the local position of the respective LB blocks in the spacetree, up to \( L + 1 \) iterations per time step may be required, assuming an adaptive LB grid with \( L \) levels. This worst-case-scenario, however, may only occur very rarely. In average, two iterations per refinement/coarsening are sufficient. These iterations are extremely cheap compared to the LB solver iterations, as the refinement mapping only implies work on the Peano vertices; it does not require any modification of the LB data on all Peano vertices, except for those that are created or deleted.

It may further be noted that the triggering of the additional traversals for Peano vertex creation and removal is not required in each (coarsest grid) LB time step. Consider all scenarios for which AMR-LB schemes have been used so far—phase-tracking in multiphase simulations, velocity and density field tracking or fluid-structure interaction problems where AMR is used to dynamically resolve the moving geometry. In these scenarios, the speed of the respective information on the grid is strictly limited by the speed of sound \( c_s \) of the LBM. Hence, triggering the refinement every \( 1/c_s \approx 1.73 \) time steps is sufficient\(^{18} \). At the first glance, applying the refinement only two out of three LB time steps does not look like a severe improvement. However, especially for geometry-induced tracking, where the geometry typically cannot move significantly faster then the maximum flow velocity, an improved estimate for the refinement and coarsening intervals can be given. Let the maximum velocity of the moving geometry be defined as \( u_{\text{max}} := C \cdot c_s \) where the constant \( C \) is typically of the order of the Mach number, \( C \in O(\text{Ma}) \). The maximum dimensionless time that it takes until the geometry has traversed one LB cell can thus be approximated by \( dt^{\text{geo}} = 1/u_{\text{max}} \approx 1.73/C \). Using the low Mach number assumption in LB schemes and hence choosing the Mach number of order 0.1, we can also expect the constant \( C \) to be of same magnitude. With \( C \approx 0.1 \), the time for the geometry to traverse a LB cell arises to be \( dt^{\text{geo}} \approx 17.3 \). Thus, updating the grid every 17 time steps is sufficient for this scenario. For strongly diffusive scenarios, the maximum velocities are often much smaller than in this example, so that the dynamic refinement procedure becomes even cheaper.

### 11 Navier-Stokes Simulations within Peano

Besides the Lattice Boltzmann-based adaptive implementations, former works focussed on the development and efficient incorporation of a Navier-Stokes solver into Peano\(^{133} \). The respective solver which is included in the Peano framework, Version 1\(^{10} \), allows for continuum flow simulations on adaptive grids in two and three dimensions. It is also based on the pressure iteration scheme from Sec. 6.2, using different space (finite elements, interpolated differential operators) and time discretisation schemes (explicit Euler, Crank-Nicolson, etc.). Although first experiments in the context of coupled Lattice Boltzmann–Navier-Stokes simulations have been performed in Peano V. 1 (see for example\(^{134} \)), major parts of the current work arose from simulations within Peano, V. 2. Within the scope of the DiParTS-project\(^{20} \) and related works\(^{92, 93} \), a new finite difference Navier-Stokes solver was implemented within Peano V. 2, based on the descriptions in\(^{70} \). Thus, if not mentioned otherwise, the results from Sec. 16.2, which involve a Navier-Stokes description of the flow, are based on the latter implementation. In the following, the algorithmic realisation of the numerical scheme from Sec. 6.2 within the Peano framework is shortly described.

\(^{18}\) At this stage and in the following, the considerations are carried out in the dimensionless form where \( dt = 1 \), \( dx = 1 \) and \( c_s = 1/\sqrt{3} \).

\(^{10}\) Although not officially declared to be split into two versions, Peano, Version 1, refer to the original framework implementation which is described in\(^{178} \). Peano Version 2 refers to the refactored Peano code, developed since 2010 at the Chair of Scientific Computing, Technische Universität München.

\(^{20}\) http://www5.in.tum.de/peano/projects/diparts/index.html

82
As required by the finite difference scheme from [70], the pressure and velocity values need to be stored on a staggered grid, with the velocities logically related to the midpoint of the cell’s edges in case of two-dimensional simulations. Peano’s data structures, however, do not model faces or edges separately, but only provide vertex and cell structures. Therefore, the edge data are stored within the Peano cell structure. Besides, velocity and pressure values from one cell need to be provided to its direct neighbours during the evaluation of the finite difference expressions. These expressions only involve velocities and pressures from the neighbouring cell edges or cell centres, respectively. For this purpose, averaged values of these quantities are stored on the Peano vertices; an example for the velocity component $u_0$ is shown in Fig. 11.1. With the average values and local quantities available within each Peano cell, all finite difference expressions can be evaluated locally, using the callback-mechanism of Peano. The single algorithmic phases to solve the Navier-Stokes system—setting boundary conditions for the velocity field, assembling the right hand side of the pressure Poisson equation, solving the Poisson equation and carrying out the time integration for the velocity field—are implemented in separate mappings. For solving the Poisson equation, a straightforward SOR solver is currently used, following the approach from [70]. The respective Peano code snippet for solving the Navier-Stokes problem is given below:

```java
double time = 0;
repository.getSpacetreeGridState().setCurrentTime(time);
while(time < t_end){
  // set boundary conditions
  repository.switchToSetVelocitiesBoundary(); repository.iterate();
  repository.switchToSetScenarioBoundary(); repository.iterate();
  // computation of the right hand side
  repository.switchToComputeVelocitiesDerivatives();
  repository.iterate();
  repository.switchToComputeRightHandSide(); repository.iterate();
  // solve pressure Poisson equation
  int it = 0;
  repository.getSpacetreeGridState().setResidual(1.0+eps);
  while( (it<itermax) && (repository.getSpacetreeGridState().getResidual()>eps)){
    repository.switchToSORStep(); repository.iterate();
    repository.switchToComputeResidualNormAndSetPressureBoundary();
  }
}
```
12 Lattice Boltzmann–Navier-Stokes Simulations

12.1 Motivation

Although Lattice Boltzmann methods are considered to be mesoscale methods, they are often used in the context of macroscopic continuum flow simulations. The commercial software Exa PowerFLOW\(^\text{21}\) has become an established alternative to other CFD softwares in the field of numerical aerodynamics for automotive engineering. Different comparisons between Navier-Stokes- and Lattice Boltzmann-based solvers have been drawn in [63, 125, 174]. Amongst others, Geller et al. showed the superiority of (implicit) Navier-Stokes schemes for stationary laminar flow problems [63] in terms of runtime based on their simulation results for flows through sphere packs. They also pointed out the difficulties in comparing the two approaches—modelling errors (for complex fluids), discretisation errors and the efficient implementation of adaptive methods have a major influence, rendering the comparison of both approaches a very difficult task. Still, from these observations and the features and limitations of both approaches (see Sec. 5.8 and 6.4), it becomes apparent that for particular flow scenarios the one or the other method may provide significant advantages in terms of discretisation, modelling or serial or parallel runtimes.

Not every scenario can be assigned to one of the groups of “Navier-Stokes’ or Lattice Boltzmann’s favourable flow problems”. Consider the geometry in Fig. 12.1, a branched microchannel system with a maximum cross-section ratio 1:48.6. Let’s further assume a stationary flow in the system and a Knudsen number \( Kn = 0.005 \) in the coarsest channel. According to Tab. 5.1, the region in the coarsest channel can still be treated by a standard continuum solver. Determining the local Knudsen number for the finest channel yields \( Kn = 0.243 \). Slip and transition flow is therefore expected in the small-sized channels. In these channels, a particular simulation method which accounts for the respective physical effects (cf. Sec. 5.7) is required. Hence, referring to the argumentation from above, none of the two

---

\(^{21}\)http://www.exa.com
methods—Navier-Stokes or Lattice Boltzmann—might represent the perfect choice with respect to performance and physical modelling. One approach consists in applying enhanced Lattice Boltzmann techniques in the small-sized channels and solving the big channel with efficient (multigrid-based) Navier-Stokes solvers. A respective example is to be picked up in Chap. 15 in terms of enhanced, spatially adaptive Lattice Boltzmann simulations.

Another problem may arise when the flow problem under consideration requires very long time intervals to be simulated. The explicit time stepping in the Lattice Boltzmann scheme with its tiny time steps hence poses a severe bottleneck in this case. A respective flow scenario—a long-time simulation of particle transport in a nanopore—is described in Sec. 16.2 where coarse-grained Navier-Stokes and fine-scale Lattice Boltzmann simulations are combined into a hybrid approach.

Following this discussion, a coupling of Navier-Stokes and Lattice Boltzmann methods represents a possible approach to overcome the drawbacks of each of the two methods. In the following, a strategy for spatially coupling Lattice Boltzmann and Navier-Stokes solvers is developed. Its fundamentals are discussed in Sec. 12.2. Remarks on implementational concepts within the Peano framework and coupling strategies are presented in Sec. 12.3.

### 12.2 Exchange of Flow Quantities: An Optimisation Approach

Several spatial couplings for Lattice Boltzmann automata and differential equation systems have been proposed in earlier publications [4, 5, 110, 112]; within this context, only the works by Latt et al. [108, 110] address the particular coupling of the Lattice Boltzmann method and a finite difference incompressible Navier-Stokes solver for fluid dynamics purposes.

Referring to the general theory on the relation between Navier-Stokes and Lattice Boltzmann methods, i.e. the Chapman-Enskog expansion (cf. Sec. 7.2), it should be remarked that one obtains the general Navier-Stokes system which also contains compressibility effects. A direct coupling of the (weakly) compressible Lattice Boltzmann system and the purely incompressible Navier-Stokes solver hence may seem inconsistent at a first glance—although Mach numbers are required to be small in the derivation of the Lattice Boltzmann scheme, there will still be a compressibility error occurring in the simulations. In the following, this error is assumed to be very small and thus negligible. This can be achieved by taking into account the Mach number in the Lattice Boltzmann system. Since the Mach number can be explicitly determined in Lattice Boltzmann simulations, cf. Sec. (5.2), it needs to be chosen small enough, so that the compressibility effects do not significantly contribute to the overall simulation outcome.\(^{22}\)

Let the computational domain \(\Omega\) be divided into two sub-domains \(\Omega = \Omega^{LB} \cup \Omega^{NS}\) where the Lattice Boltzmann method is applied in \(\Omega^{LB}\) and the Navier-Stokes solver is applied in \(\Omega^{NS}\), respectively. The domains \(\Omega^{LB}\) and \(\Omega^{NS}\) overlap by at least \(dx = \max\{dx^{LB}, dx^{NS}\}\) where \(dx^{LB}\), \(dx^{NS}\) are the mesh sizes of the Lattice Boltzmann and the Navier-Stokes solver. This overlap is required to allow for interpolation of the fluid quantities from the Navier-Stokes to the LB grid and vice versa.

A spatial coupling requires a two-way exchange of quantities between the Lattice Boltzmann and the Navier-Stokes domain: on the one hand, the velocities need to be derived from the particle distributions and imposed to the Navier-Stokes solver as boundary conditions.\(^{23}\)

On the other hand, the pressure and velocity values from the Navier-Stokes solver need to be used to construct a valid set of particle distributions that serves as boundary condition on Lattice Boltzmann side. Going from Lattice Boltzmann to Navier-Stokes is hence straightforward: given a set of particle distributions, the average flow velocity in this Lattice Boltzmann cell can be evaluated via Eq. (5.7) and be imposed to the Navier-Stokes solver. Depending on the underlying spatial discretisation, interpolation of the LB velocities onto the respective Navier-Stokes grid is necessary. Based on the block-structured Lattice Boltzmann solver from Sec. 10.3 and the Navier-Stokes implementation described in Chap. 11,\(^{24}\)

\(^{22}\)It may be noted that the Mach number is typically already very small in most standard Lattice Boltzmann setups and hence of the order \(O(0.05)\) or even smaller.

\(^{23}\)Considering the Navier-Stokes algorithm from Sec. 6.2, it becomes clear that the pressure does not need to be mapped from Lattice Boltzmann to Navier-Stokes. It will automatically be constructed via the implicit pressure Poisson update from the velocity values.
the interpolation is accomplished as follows: $2^D$ LB cells are identified which surround
the location on the Navier-Stokes grid—that is the midpoint of a cell edge in the two-dimensional
case—where the Navier-Stokes velocity shall be prescribed. The velocity component of
the Navier-Stokes velocity $u^D_{NS}$ is then determined by averaging of the LB velocities $u_{i\alpha}^{LB}$ of all
cells $c$:

$$u^D_{NS} = \frac{1}{2^D} \sum_{c=1}^{2^D} u_{i\alpha}^{LB}$$  \hspace{1cm} (12.1)$$

Going from Navier-Stokes to Lattice Boltzmann is more challenging: given $D+1$ unknowns,
that is pressure and fluid velocities, and $D(D+1)/2$ values for the fluid-internal stresses
(see Eq. (6.41)) on the Navier-Stokes side, $Q$ distributions need to be constructed. Latt et
al. propose to first split the distributions $f_i$ into equilibrium and non-equilibrium part [110],
$f_i = f_i^{eq} + f_i^{neq}$, and construct both parts separately. Especially for the non-equilibrium
terms, they propose several simplifications in the asymptotic analysis, such as a restriction to
an equilibrium function where the quadratic terms in the fluid velocity are omitted (see Eq.
(5.10) for the expression of the equilibrium function) or the assumption of having steady-
state conditions for the non-equilibrium parts ($\partial f_i^{neq}/\partial t \approx 0$). In the following, a different
method is developed to construct the distributions $f_i$. Major parts of the underlying concept
have been previously published in [134]. No particular approximations with respect to the
asymptotic behaviour are necessary in this case. Some steps, however, are carried out
analogously as proposed by Latt et al. in [108, 110].

In a first step, the distributions are split as discussed in [110]: $f_i = f_i^{eq}(\rho, u) + f_i^{neq}$.
Given the fluid pressure—and hence the fluid density in the Lattice Boltzmann sense—and
the velocity at the midpoint of a Lattice Boltzmann cell, the equilibrium part $f_i^{eq}$ can be
constructed. As the average fluid density is scaled to unity in the dimensionless case for
Lattice Boltzmann simulations, a shift of the fluid density by a respective offset is required.
This step is valid since it is only the pressure gradient that plays a significant role in Navier-
Stokes simulations; the arising fluid density $\rho = p/c_s^2$ can hence be shifted by arbitrary
offsets. A possible formula for the shift is given in [110] and is used in the following:

$$\rho = \left( \frac{p - p_{\text{GSNS}}}{c_s^2} + 1 \right)$$  \hspace{1cm} (12.2)$$

where $p$ denotes the pressure at the respective LB cell’s midpoint and $p_{\text{GSNS}}$ is the average
pressure value computed from all NS cells on the Lattice Boltzmann–Navier-Stokes bound-
ary. Alternatively, $p_{\text{GSNS}}$ can be chosen to be a known constant in Navier-Stokes simulations
where the pressure is kept fixed via Dirichlet conditions. Similar to the transition from Lattice
Boltzmann to Navier-Stokes, the determination of the pressure and the fluid velocity in
the centres of the LB cells requires interpolation techniques. In order to provide a sufficient
level of accuracy in the boundary regions, second-order spatial interpolation is applied in
both cases.

With the equilibrium parts constructed from the degrees of freedom of the Navier-Stokes
system, the construction of the non-equilibrium parts is addressed next. It is still an open
issue how to choose the non-equilibrium parts in a suitable way; however, some constraints
are given in terms of the lower-order moments. The zero- and first-order moments of
the non-equilibrium parts need to vanish, that is $\sum_i f_i^{neq} = \sum_i f_i^{neq} c_i = 0$, as mass and momentum
are completely prescribed via the equilibrium state. Another major constraint can be
derived from the Chapman-Enskog analysis, see Sec. 7.2: the second-order moments of the
non-equilibrium parts need to relate to the viscous stress tensor of the fluid. For the BGK col-
lision rule, this relation is given by Eq. (7.79). Hence, $1 + D + D(D+1)/2 = (D+1)(D+2)/2$
equations for the non-equilibrium parts $f_i^{neq}$, $i = 1, \ldots, Q$, are given. As it holds that
$Q > 1 + D + D(D+1)/2 = (D+1)(D+2)/2$ for typical Lattice Boltzmann discretisations
in the hydrodynamic case, more degrees of freedom are left on the Lattice Boltzmann side.

The non-equilibrium parts are consequently not defined uniquely by the mass, momentum
and stress constraints.

Different possibilities are possible to fix the remaining degrees of freedom. One approach
consists in considering the space of the higher-order moments. Fixing all higher-order moments, for example setting all higher-order moments to zero, would immediately lead to a closed description for the non-equilibrium parts \( f_i^{\text{neq}} \). An open question that is part of current research for many moment-based approaches in Lattice Boltzmann methods also remains in this case: how should the higher-order moments be fixed to yield an optimal approximation to the underlying flow problem? And in which sense shall the respective solution be chosen “optimal”? The latter question yields the solution strategy that is presented in the following [134]. Instead of fixing the higher-order moments, we search for a non-equilibrium vector \( f^{\text{neq}} \in \mathbb{R}^Q \) that yields a minimum of a function \( g(f^{\text{neq}}) : \mathbb{R}^Q \to \mathbb{R} \) such that mass, momentum and stress constraints are fulfilled. Therefore, the following optimisation problem needs to be solved:

\[
\min_{f^{\text{neq}} \in \mathbb{R}^Q} g(f^{\text{neq}}) \quad \text{such that}
\]

\[
\sum_i f_i^{\text{neq}} = 0
\]

\[
\sum_i f_i^{\text{neq}} c_{i,\alpha} = 0 \quad \forall \alpha \in \{1, ..., D\}
\]

\[
\sum_i f_i^{\text{neq}} c_{i,\alpha} c_{i,\beta} = -c^2 \tau \left( \partial_{x_\alpha} u_\alpha + \partial_{x_\beta} u_\beta \right) \quad \forall \alpha, \beta \in \{1, ..., D\}
\]

where the latter constraint arises from Eqs. (6.41) and (7.79) in the incompressible limit and assuming a dimensionless fluid density \( \rho = 1 \). In the following, the side constraints are denoted as linear system \( A \cdot f^{\text{neq}} = b \) with matrix \( A \in \mathbb{R}^{(D+1)(D+2)/2 \times Q} \) and right hand side \( b \in \mathbb{R}^{(D+1)(D+2)/2} \).

Any solution to the given minimisation problem will fulfill the macroscopic relations for mass, momentum and stresses due to the side constraints. Besides, the non-equilibrium terms can be “controlled” via the function \( g(f^{\text{neq}}) \) so that they are locally kept small in a certain sense. However, not every arbitrary function \( g \) can be used since it needs to be guaranteed that there exists a (unique) solution to the minimisation problem for all possible macroscopic states and respective constraints, that is for all right hand sides of the side constraints.

One possible class of functions for \( g(f^{\text{neq}}) \) is given by second-order polynomials \( g(f^{\text{neq}}) := \sum_{i \leq j} g_{ij} f_i^{\text{neq}} f_j^{\text{neq}} + \sum_i g_i f_i^{\text{neq}} + g_c \), with \( g_{ij}, g_i, g_c \in \mathbb{R} \). If it additionally holds \( g_{ii} > 0 \) for all \( i = 1, ..., Q \), the solution of the minimisation problem from above is unique. The proof for the latter statement can be accomplished by considering the underlying Lagrange formulation of the optimisation problem; it can be found in the appendix of [134]. The optimisation problem can be solved via determining the Lagrange multipliers \( \lambda \in \mathbb{R}^{(D+1)(D+2)/2} \) from a linear system \( ADA^\top \lambda = r \), with the components of \( r \in \mathbb{R}^{(D+1)(D+2)/2} \) defined as

\[
r_k = b_k + \sum_i \frac{A_{ki}}{2g_{ii}} \left( g_i + \sum_{j=1}^{i-1} g_{ji} + \sum_{j=i+1}^{Q} g_{ij} \right)
\]

and a diagonal matrix \( D = \text{diag}(1/(2g_{11}), ..., 1/(2g_{QQ})) \). The non-equilibrium parts \( f_i^{\text{neq}} \) arise from the Langrange multipliers:

\[
f_i^{\text{neq}} = \frac{1}{2g_{ii}} \left( \sum_k \lambda_k A_{ki} - g_i - \sum_{j=1}^{i-1} g_{ji} - \sum_{j=i+1}^{Q} g_{ij} \right).
\]

Different respective polynomials may represent a reasonable choice. In the following, the discussions are restricted to polynomials where \( g_{ij} = 0 \) for \( i \neq j \) as well as \( g_i = g_c = 0 \). This further implies the rule \( f^{\text{neq}} = DA^\top (ADA^\top)^{-1} b \) for the non-equilibrium parts. As the first \( D+1 \) entries of the vector \( b \) only contain zeros due to mass and momentum conservation, it is sufficient to only apply the \( D(D+1)/2 \) last columns of the respective matrix to the \( D(D+1)/2 \) last entries of \( b \). The arising reduced matrix for non-equilibrium construction is denoted by

\[
B^{\text{reim}} := \left( DA^\top (ADA^\top)^{-1} \right)_{i=1, j=D+2}^{i=Q, j=(D+1)(D+2)/2} \in \mathbb{R}^{Q \times D(D+1)/2}
\]
and consequently only operates on the viscous stress terms.
Examples for the optimisation polynomial \( g(f^{\text{neq}}) \) comprise:

- **Squared L2-norm:** \( g_{L^2}(f^{\text{neq}}) := \sum_i f_i^{\text{neq}}^2 \)
  Minimising the \( L_2 \)-norm represents one of the standard choices in optimisation. As the coefficients \( g_{ii} = 1, \ g_{i} = 0 \ \forall i, \ g_{ij} = 0 \ \forall i \neq j, \ g_{c} = 0 \) are constant, the matrix \( ADA^T \) is known a priori and can hence be inverted before the coupled simulation. The determination of the non-equilibrium parts hence degenerates to a simple and cheap local matrix-vector operation.

- **Squared Knudsen norm:** \( g_{Kn}(f^{\text{neq}}) := \sum_i \left( \frac{f_i^{\text{neq}}}{f_{i}^{\text{eq}}(1,0)} \right)^2 \)
  The ratio of non-equilibrium and equilibrium parts can be taken as a measure for the Knudsen number\(^{24}\). Minimising the squared Knudsen-norm hence can be interpreted as minimising the Knudsen number and hence staying as close to the continuum regime as possible. Since the squared Knudsen-norm depends on the local flow quantities, the optimisation problem needs to be solved in each LB cell of the Lattice Boltzmann–Navier-Stokes interface, yielding significantly higher computational costs as in the case of the \( L_2 \)-norm.

- **Approx. squared Knudsen norm:** \( g_{aKn}(f^{\text{neq}}) := \sum_i \left( \frac{f_i^{\text{neq}}}{f_{i}^{\text{eq}}(1,0)} \right)^2 = \sum_i \left( \frac{f_i^{\text{neq}}}{c_i} \right)^2 \)
  The approx. squared Knudsen-norm is derived from the squared Knudsen-norm in the constant-density-zero-velocity limit. It is expected to perform similar to the squared Knudsen-norm, but has the advantage that all coefficients of \( g_{aKn}(f^{\text{neq}}) \) are constant. The non-equilibrium parts can hence be computed from a matrix-vector multiplication as it is the case for the \( L_2 \)-norm.

Due to its low computational costs and its physical relevance, the results in Sec. 16.2 make use of the approx. squared Knudsen-norm.

With the new method introduced, a short comparison to the existing approach by Latt et al. [108, 110, 111] shall be drawn. Therefore, the construction rule for the non-equilibrium parts is considered in more detail. It is further assumed that the coefficients \( g_{ii} \) of the polynomial \( g(f^{\text{neq}}) = \sum_i g_{i}f_{i}^{\text{neq}}^2 \) shall be equal for all \( f_{i}^{\text{neq}} \) which belong to lattice velocities of same magnitude: \( g_{i} = g_{j} \) if \( ||\vec{c}_i|| = ||\vec{c}_j|| \). The arising coefficients are denoted by \( g_{||\vec{c}||} \).

The matrices for non-equilibrium construction of both approaches \( B^{\text{ext.}} \) and \( B^{\text{est.}} \) are compared for different velocity discretisation schemes (D2Q9, D3Q15, D3Q19, D3Q27). The matrices \( B^{\text{eq.}} \) and \( B^{\text{est.}} \) for the D2Q9 model are shown in Eq. (12.7):

\[
B^{\text{ext.}} = \begin{pmatrix}
B_0 - \frac{1}{12} & B_B \\
-2B_0 & 0 & B_1 \\
B_0 & \frac{1}{12} & B_0 \\
B_1 & 0 & -2B_0 \\
-2B_1 & 0 & -2B_1 \\
B_1 & 0 & -2B_0 \\
B_0 & \frac{1}{12} & B_0 \\
-2B_0 & 0 & B_1 \\
B_0 - \frac{1}{12} & B_0 \\
\end{pmatrix}
\]

\[
B^{\text{est.}} = \begin{pmatrix}
-\frac{1}{18} & -\frac{1}{12} & -\frac{1}{18} \\
\frac{1}{9} & 0 & -\frac{2}{9} \\
-\frac{1}{18} & \frac{1}{12} & -\frac{1}{18} \\
-\frac{2}{9} & 0 & \frac{1}{9} \\
\frac{4}{9} & 0 & \frac{4}{9} \\
\frac{1}{9} & 0 & \frac{1}{9} \\
-\frac{1}{18} & \frac{1}{12} & -\frac{1}{18} \\
-\frac{1}{18} & \frac{1}{12} & -\frac{1}{18} \\
-\frac{1}{18} & \frac{1}{12} & -\frac{1}{18} \\
\end{pmatrix}
\]

with

\[
B_0 = - \frac{g_1 + 2g_0}{6(g_\sqrt{2} + 4g_1 + 4g_0)}
\]

\[
B_1 = - \frac{g_\sqrt{2} + 2g_1}{6(g_\sqrt{2} + 4g_1 + 4g_0)}
\]  

(12.8)

The columns are related to the stresses \(\tau_{xx}, \tau_{xy}, \tau_{yy}\) and are sorted in the same way the stresses are listed here. The matrices for the other velocity sets are listed in the appendix. It should first be noted that both approaches operate on slightly different definitions of the stress tensor: the optimisation method from above relates the viscous stresses to \(\sum_i f_{\text{neq}}^{\text{eq}} c_\alpha c_\beta\) whereas the expression by Latt et al. is based on a traceless form [111]:

\[
f_i^{\text{neq}} = -\frac{w_i \tau}{c_s^2} \rho \sum_{\alpha,\beta} (c_\alpha c_\beta - c_\alpha^2 \delta_{\alpha\beta}) \partial_{x_\beta} u_\alpha.
\]  

(12.9)

As a consequence, this modified expression would remove one side constraint for the diagonal stress terms \(\sum_i f_i^{\text{neq}} c_\alpha c_\alpha\). The respective columns (column 1 and 3 in the D2Q9 case) in the two matrices \(B^{\text{opt}}\) and \(B^{\text{Latt}}\) are therefore expected to behave differently in the sense that the optimisation approach cannot exactly capture the Latt approach on the one hand, but still provides tunable parameters in form of the polynomial coefficients on the other hand. The structure of these columns, however, is exactly the same in both cases. For the non-diagonal stresses, it turns out that the approach by Latt et al. can be exactly recovered by the optimisation approach: in the case of velocity sets with three different velocity magnitudes (D2Q9, D3Q15, D3Q19), the respective columns in the matrices \(B^{\text{opt}}\) and \(B^{\text{Latt}}\) are identical, independent from the exact choice of the isotropic polynomial coefficients \(g\|c_i\|\). For the D3Q27 model, one more degree of freedom consists in the choice of the coefficients \(g\|c_i\|, \beta\), since four different velocity magnitudes are included in the model. The columns related to the non-diagonal viscous stress terms \(\sum_i f_i^{\text{neq}} c_\alpha c_\beta, \alpha \neq \beta\), are therefore not necessarily identical. Still, the identity can be reached by choosing \(g_\sqrt{3} = 4g_\sqrt{2}\).

12.3 Coupled Implementations within Peano

12.3.1 Technical Details

In the following, the methodology of coupling two components within Peano is described. This is important in order to facilitate the process of integrating the Lattice Boltzmann application into the Navier-Stokes solver for the hybrid simulations. Several aspects on Peano-internal application coupling have been described in depth in [134] with respect to Peano, V. 1. No particular descriptions have been made for Peano, V. 2, yet. Peano, V. 2, comes with a very clear concept with regard to the application layout. Typical differences to Peano, V. 1, include the usage of a global state object or an adapter repository. Although these ideas are not necessarily new per se, their identical realisation in all applications enforce a very specific structure of all built-in applications. On the one hand, this is advantageous since the interfaces against which the Peano kernel can for example operate are formulated clearly and therefore allow to outsource even more general functionality to the kernel. On the other hand, the application layout becomes more stringent. This yields less flexibility for Peano-internal application couplings. As a consequence, the template-based coupling concept described in [134] cannot be directly applied to applications in Peano, V. 2. An alternative for Peano, V. 2, is therefore described within this section and follows the steps of code development from Fig. 9.3.

A coupled application combines two or more Peano applications and builds a separate application within Peano. Following the PeProt-based modelling, common data structures for the state, vertices and cells need to be set up. In case that the complete description of a given type—that is vertex, cell or state—shall be reused, the easiest way is to define the new type via inheriting from the original application-bound DaStGen definitions. The hierarchy flattening which is automatically performed by DaStGen can cause problems if two distinct variables have identical names in both applications. The same holds for respective
method definitions. These conflicts need to be resolved manually by renaming the variables or methods. Afterwards, all required cell and vertex data are available everywhere on the Peano grid, that is each Peano cell and vertex holds all the data defined in the merged DaStGen description. Next, adapters and mappings of the coupled implementation need to be declared via PeProt scripting. Depending on the coupling, the mapping and adapter declarations of the coupled application may comprise all declarations or only single aspects of the original applications. Besides, additional coupling-specific mapping and adapter definitions are typically required. A direct combination of the original applications’ PeProt scripts is therefore not feasible. Still, the required mapping and adapter definitions can be extracted from the original scripts and used in the new coupling application script. From the DaStGen- and PeProt-based specifications, the coupled application’s template can be created.

With the vertex, cell, state and mapping classes generated, the functionality from the original applications needs to be ported to the coupled application. The mappings should not contain specific functionality, but only forward respective calls to functional classes. A particular problem arises from the definition of the classes in the original applications which hold the specific functionality \textit{(functional classes)}: assume a given functional class AccumulateVertexCounter shall enumerate the vertices. A respective implementation from the original application is already given. This implementation can be accomplished in two different ways:

1. The vertex is directly handed over to an instance of the AccumulateVertexCounter from within the respective mapping:

   ```cpp
   void MyMapping::touchVertexFirstTime(OriginalAppVertex & vertex){
       _accumulateVertexCounter.setVertexId(vertex);
   }
   
   void AccumulateVertexCounter::setVertexId(OriginalAppVertex & v){
       int id = _counter;
       v.setId(id);
       _counter++;
   }
   ```

2. The required data is extracted from the vertex in the mapping, handed over to the instance of the AccumulateVertexCounter and finally written back ("gather-scatter"-like operations):

   ```cpp
   void MyMapping::touchVertexFirstTime(OriginalAppVertex & vertex){
       int id = vertex.getId();
       _accumulateVertexCounter.setVertexId(id);
       vertex.setId(id);
   }
   
   void AccumulateVertexCounter::setVertexId(int & id){
       id = _counter;
       _counter++;
   }
   ```

In principle, both approaches apply the same functionality, but the second approach is advantageous: it can be immediately applied in the coupled implementation since the AccumulateVertexCounter is completely independent from the underlying vertex type. An alternative approach lies in overloading or templatising the respective method with respect to vertex, cell and state types. This methodology is used besides the gather-scatter approach in Peano, V. 1. However, it either yields code duplications or overheads during compiling. In order to reuse functionality from existing mappings in other contexts, their callbacks should hence be structured as follows:

1. Extraction: extract primitive types from vertex, cell or state objects
2. Execution: pass references of the primitives to functional object’s method
3. Storage: write back primitives to vertex, cell and state objects

This methodology allows to reuse the whole functionality of all applications. Still, the extraction of the primitives needs to be established and incorporated into the mappings. At the current stage, this step needs to be carried out manually. For more complex coupling scenarios, a respective implementation may not be feasible anymore. An extension of PeProt or the creation of a coupling tool which automatically generates the code for extraction, execution and storage might therefore be desirable in future. Assuming a dependence of the respective tool on DaStGen and PeProt, this could look as follows for the example from above:

```plaintext
// include PeProt script
include myapplication.peano-specification

// for a mapping defined in myapplication.peano-specification,
// define the extraction, execution and storage operations
MyMapping:
  functional-class: AccumulateVertexCounter
  touchVertexFirstTime:
    extract: vertex.vertexId
    execute: AccumulateVertexCounter
    store: vertex.vertexId
```

In the present case, the Lattice Boltzmann application has been integrated into the Navier-Stokes solver application. The functional units ported from the Lattice Boltzmann application to the coupled Navier-Stokes–Lattice Boltzmann application comprise the Lattice Boltzmann solver on regularly refined, that is non-adaptive, grids and the moving obstacle handling. Additional functionality is required for the transition from one solver to the other: interpolation classes are implemented to map velocity and pressure values from Navier-Stokes to Lattice Boltzmann and vice versa, as well as a class to initialise the particle distribution functions according to the optimisation approach from Sec. 12.2. With the Lattice Boltzmann solver working on embedded block-structured grids and meant to operate in the small-scale regime, it is assumed that the Lattice Boltzmann solver works on finer grids than the Navier-Stokes solver. The user can define a resolution $dx$ for the Lattice Boltzmann solver and a refinement region $[x_0, y_0] \times \ldots \times [x_D, y_D]$ via the configuration. Depending on the scenario of interest, the Navier-Stokes domain is refined inside this region until the Lattice Boltzmann resolution is reached. Within this refined region, the Lattice Boltzmann solver can then be applied. In the following, two different hybrid schemes are discussed: a two-way coupling scheme which consistently couples Navier-Stokes and Lattice Boltzmann flow fields in both directions and a one-way coupling approach for particle transport simulations.

### 12.3.2 Two-Way Flow Coupling

In order to address flow problems such as the one from Fig. 12.1, a decomposition of the flow domain into a Lattice Boltzmann and a Navier-Stokes region is desirable. A respective two-way coupling approach is shown in Fig. 12.2. The Lattice Boltzmann region is embedded into the Navier-Stokes region. The inner LB cells are coloured red. The green LB cells denote the cells in which the mapping from Navier-Stokes to Lattice Boltzmann is carried out. The Navier-Stokes region covers the entire computational domain. All components of the velocity vector that are stored at one of the light blue-coloured locations are obtained from the Lattice Boltzmann simulation.

The algorithm works as follows: starting from the Navier-Stokes solver, the average pressure $p_{\text{NS}}$ over the overlap region (green NS cells) is determined. The pressure and velocity values as well as the stress tensor are determined in the outermost Lattice Boltzmann cells using second-order interpolation and analytical differentiation of the velocity field. The respective LB cells are located on the hanging nodes and are neighboured to the first layer of
Figure 12.2: Two-way flow coupling of a fine-scale Lattice Boltzmann simulation and a coarse-grained Navier-Stokes solver. The small green-coloured Lattice Boltzmann cells are initialised in each LB time step based on the pressure-, momentum- and stress-related optimisation strategy from Sec. 12.2. After LB time stepping is finished, the flow velocity from the Lattice Boltzmann simulation is interpolated at the positions denoted by light blue-coloured circles. The interpolated values are then set on the Navier-Stokes grid. Lattice Boltzmann cells on the persistent Peano vertices. From the interpolated values, the particle distribution functions are constructed according to the descriptions in Sec. 12.2. Now, $N$ time steps are carried out in the Lattice Boltzmann domain. In each time step, the boundary conditions prescribed by the Navier-Stokes solver are set on the hanging node boundaries. The number of time steps $N$ is currently chosen such that $N \cdot dt^{LB} = dt^{NS}$ where $dt^{LB}, dt^{NS}$ denote the time steps of the Lattice Boltzmann and the Navier-Stokes simulation. Choosing a bigger value of $N$ may also be feasible in the stationary case. After the Lattice Boltzmann time steps, the velocity values in the Lattice Boltzmann domain are coarse-grained and written to the Navier-Stokes simulation following Eq. (12.1). The coarse-graining is currently activated for the whole inner part of the Lattice Boltzmann domain. In order to stay divergence-free, the discrete geometry representation needs to be consistent close to the overlap region in both Navier-Stokes and Lattice Boltzmann simulation. Therefore, consider Fig. 12.3 on the right: the two obstacles yield no-slip boundary conditions for the Navier-Stokes solver. In contrast, the Lattice Boltzmann simulation can still capture flow between the obstacles due to the finer mesh. The interpolation from Navier-Stokes onto the Lattice Boltzmann grid will, however, suppress any flow in this region. The pressure does not need to be mapped from Lattice Boltzmann to Navier-Stokes. Since the Navier-Stokes velocity vectors are “corrected” by the fine-scale Lattice Boltzmann method, a consistent pressure is obtained in the next Navier-Stokes solver iteration. This iteration is carried out on all Navier-Stokes cells including the green- and red-coloured cells from Fig. 12.2.

12.3.3 A Hierarchical Hybrid Particle Transport Method

As pointed out in the motivation for hybrid Lattice Boltzmann–Navier-Stokes simulations, Lattice Boltzmann methods typically suffer from very small time steps due to their explicit nature. For the simulation of very long time intervals, a huge number of Lattice Boltzmann time steps is hence required. Different approaches can be used to reduce computational costs for respective flow solvers:

1. Switch to an implicit scheme. A multitude of implicit Navier-Stokes schemes is available and has been discussed in the literature. However, if the mesoscopic nature of Lattice Boltzmann is required to capture the correct flow physics, an implicit Navier-Stokes solver will not be the cure.
Figure 12.3: Consistent vs. inconsistent geometry description in two-way coupled Navier-Stokes–Lattice Boltzmann simulations. The Navier-Stokes solver is applied in the left region on the black circles, and the Lattice Boltzmann solver is used on the fine grid. Geometrical obstacles (grey-coloured regions) cover parts of the grid. Left: consistent flow geometry. One layer of blue-coloured circles, that is x- and y-velocity components, exist inside the Lattice Boltzmann region. The interpolation will hence yield correct velocity values for Navier-Stokes at these points. The same holds when going from Navier-Stokes to Lattice Boltzmann. Right: inconsistent flow geometry. In the fine-scale representation of the geometry, fluid flow is allowed between the two obstacles. This, however, is not the case in the coarse-grained Navier-Stokes representation. The interpolation from Navier-Stokes to Lattice Boltzmann will hence always yield zero flow velocity in the narrow region.

2. Use a coarse-grained flow simulation in spatial regions where less accuracy is required and hence allow for locally coarser time steps in these regions. Spatially adaptive Lattice Boltzmann simulations based on velocity-scaling between the grid levels immediately imply both, spatial coarse-graining and local coarse time steps, cf. Sec. 5.4. Using a Navier-Stokes solver on the coarser levels instead of applying Lattice Boltzmann can also represent an alternative in case that the mesoscopic nature is not required in the respective regions.

3. Use a strongly time-adaptive method. If large time steps can be used for time intervals which are of less importance in the current simulation, a considerable amount of time can be saved.

4. A combination of the approaches 1.-3.

In the following, a method is sketched which combines Lattice Boltzmann and Navier-Stokes solvers and incorporates the second and third approach into a unified scheme. For this purpose, particle transport is considered; respective long-time simulations of physical significance from meso- to macroscale are described in more detail in Sec. 16.2. Coming back to the software environment Peano for Lattice Boltzmann and Navier-Stokes simulations, Peano’s grid structure does not only yield adaptive grids as discussed so far, but in fact delivers a full hierarchy of Cartesian grids. This allows to have a representation of the fluid dynamics on each grid level. Assume for example a regularly refined two-dimensional grid which covers the unit square and consists of $81 \times 81$ cells. If the coarsest grid under consideration is chosen to have $9 \times 9$ cells, three different flow representations can be embedded in the tree structure. The coarsest grid will typically not deliver very accurate results due to the very coarse grid representation. Still, the solution might be sufficient if a particular time interval is of less significance for the evolution of the current simulation. As long as an expert can tell the simulation when to use a finer or coarser grid resolution and thus a more or less accurate flow description, the interesting flow characteristics can be captured.
This expert can either sit in front of the computer (“Computational Steering”) or—as in the present work—can be represented by an automatic error estimator which is referred to as expert system in the following. Based on the current and previous time steps, this estimator needs to decide which resolution and which model is sufficient at the current stage. The abstract algorithm 12.1 reads similar to AMR scenarios. The simulation sim is started using an initial flow model on a given grid resolution. In each time step, the expert system investigates the simulation data and decides whether to adapt the current resolution and the current model.

Algorithm 12.1 Model and grid refinement using an expert system.

\[
\begin{align*}
    \text{define time step } & \ dt, \text{ time } t, \text{ model } m, \text{ simulation } \text{sim}, \text{ initial grid resolution } h \\
    m & \leftarrow \text{ExpertSystem} \\
    \text{sim} & \leftarrow \text{Simulation}(h, m) \\
    dt & \leftarrow \text{sim} \\
    \text{for } t = 0; t < t_{\text{end}}; t = t + dt \text{ do} \\
    & \text{sim.solve}(dt, m) \\
    & \text{Model } m_{\text{new}} \leftarrow \text{ExpertSystem.decide(} \text{sim} \text{)} \\
    & \text{if } m_{\text{new}} \neq m \text{ then} \\
    & \quad \text{sim.adaptToNewResolutionAndModel}(m_{\text{new}}) \\
    & \quad \text{sim} \leftarrow m_{\text{new}} \\
    \text{end if} \\
    & \quad dt \leftarrow \text{sim} \\
    \text{end for}
\end{align*}
\]

In the following, the flow problem under consideration consists of a spherical particle transported by the flow. Two distinct ways of simulating the particle have been described in Sec. 5.6 and Sec. 6.3: on the Navier-Stokes levels, the particle is modelled implicitly by the Faxén approach. Its trajectory and velocity are computed from the Navier-Stokes velocities and pressure values in a simple post-processing step. When switching to the Lattice Boltzmann simulation, the particle is—in contrast to the Faxén theory—explicitly resolved in the flow field. The forces are exchanged in a two-way coupling of the fluid and the particle in this case resembling a simple form of explicit fluid-structure interaction. Both boundary and initial values for the Lattice Boltzmann domain need to be provided. Therefore, the optimisation-based approach from Sec. 12.2 is applied in all respective Lattice Boltzmann cells. Since the particle is not explicitly resolved in the Navier-Stokes flow field, a modelling error is expected in case of immediate switching from the Navier-Stokes to the Lattice Boltzmann solver. To account for this deficiency, the flow field in the Lattice Boltzmann simulation is solved until the steady state for the initial problem is reached. During this phase, the position and velocity of the particle are fixed, and the boundary values around the Lattice Boltzmann domain are set via the optimisation approach. Finishing the Lattice Boltzmann simulation, only the particle data, that is position, velocity and force of the particle, are coarse-grained to the Navier-Stokes regime. The virtual Faxén particles are hence updated with the respective values from the Lattice Boltzmann simulation. Due to the difference between the implicit and explicit particle modelling, no further flow quantities are coarse-grained. The fine-scale Lattice Boltzmann domain is afterwards removed from the spacetime grid and the simulation continues on all possible Navier-Stokes levels. Further coarsening of
Navier-Stokes levels may be triggered by the expert system.

Having discussed the technical steps for switching between different grid resolutions and models, the decision on when to switch needs to be made. For this purpose, the expert system is implemented in form of a service (cf. Chap. 9). Different numerical measurements and numerical analysis techniques can be used by the expert system to determine the validity of the current simulation, ranging from classical error estimators to simple and computationally cheap switching criteria. In this thesis, the following two expert strategies are applied:

1. For a well-known flow problem, the approximate behaviour over time is expected to be known. The expert system in this case switches according to the current time from a coarse to a fine flow description and vice versa.

2. The classical Faxén theory is only applicable if the particle is far away from any kinds
Initialise simulation: Initialise NS and/or LB

NS: Simulate time interval dt

Expert system: Decide on grid/model refinement

Coarse LB and switch to NS: Coarse particle description

Initiate LB: Refine to LB resolution

Initialise distributions/particle initialisation phase

Refine from NS to LB: Refine to LB resolution

Coarse LB and switch to NS: Coarse particle description

Initialise LB domain

Modify NS: Coarse refine grid

Start simulation

Finish simulation

Simulation

Initialisation

Model and grid adaption

Finalisation

Figure 12.5: Workflow in hybrid Navier-Stokes–Lattice Boltzmann simulations for particle transport.

of boundaries. The second expert system hence switches to Lattice Boltzmann if the particle approaches a geometrical object. It switches back to Navier-Stokes if the particle’s distance to the geometry exceeds a predefined threshold. This threshold depends on the particle size and the simulation setup; it is therefore currently fixed in each respective scenario.

Both strategies are very simple. Further extensions to more complex switching criteria can be easily incorporated by exchanging the respective criterion within the expert system service. No further modifications in the code are required.

Having discussed all coupling steps, the simulation workflow of the hybrid Navier-Stokes–Lattice Boltzmann simulation for particle transport is now complete. The arising simulation system is depicted in Fig. 12.5. Simulation results based on this coupling strategy are presented in Sec. 16.2.

13 Molecular Dynamics–Lattice Boltzmann Simulations

13.1 Motivation

In the previous chapter, the coupling of the mesoscopic Lattice Boltzmann scheme with the macroscopic/continuum fluid description has been discussed. Referring to the scale separation map from the introduction, cf. Fig. 2.1, the arising hybrid methods build the connection between the centre and upper-right regions of the respective graph. Both meso- and macroscopic methods are particularly attractive when molecular effects are either negligible on the coarser length and time scale or when these effects are well-understood and can be incorporated into the mesoscopic and continuum models. Examples for the latter statement have already been addressed in Sec. 5.7 in terms of additional slip effects and viscosity reduction in finite Knudsen flows. However, there are cases in which the molecular behaviour is not well understood yet. As a consequence, coarse-grained models are not well-established yet or do not exist at all for these scenarios. Examples comprise the flow simulation through porous material [152] or other geometries which contain small scale structures of molecular size. These problems may be simulated using molecular dynamics. These simulations are, however, often computationally much too expensive when it comes
to relevant applications from biology or nanotechnology. The geometrical structures of the respective scenarios may range from molecular to micron-sized lengths. The scenarios are therefore only accessible via molecular dynamics on massively parallel platforms or even not accessible at all, remembering that the trillion atom simulation from [64] could only capture a volume of $16\mu m^3$ (see Sec. 4.3). Therefore, an alternative approach to pure molecular dynamics is required which outperforms molecular dynamics in terms of computational costs on the one hand and still provides a sufficient level of physical accuracy on the other hand. In this context, the idea to couple mesoscopic and molecular simulations evolved throughout the last years.

Stepping towards the compatibility of Lattice Boltzmann and molecular dynamics methods, benchmark computations of two-dimensional flow around an obstacle have been carried out by Horbach and Succi [87]. In their numerical experiments, they use very small LB cells which are even finer than the characteristic length scale $\sigma$ of the Lennard-Jones potential from the molecular simulations. This results in a very “low separation of scales”; Horbach and Succi therefore also state that “a hydrodynamic description goes under question” under these conditions. Still, they conclude that their findings indicate the quantitative agreement of the Lattice Boltzmann method with molecular dynamics when approaching the molecular regime. As a consequence, “there appears to be a sound ground for prospective multiscale applications based on the combined use of (multigrid) LB [9] with MD.”

Several hybrid schemes have been developed and studied over the last years, amongst others [10, 33, 43, 95, 96, 164, 180]. Werder et al. [180] couple an incompressible Navier-Stokes solver to molecular dynamics to investigate the steady-state flow past a carbon nanotube. The region close the tube is resolved by molecular dynamics whereas the rest of the computational domain is treated by the Navier-Stokes solver. Dupuis et al. [43] propose a similar coupling scheme which replaces the Navier-Stokes solver by a Lattice Boltzmann automaton. In contrast to these steady-state couplings, Buscalioni et al. [33] published a time-dependent coupling of molecular dynamics and a compressible Navier-Stokes solver. Barsky et al. [10] use an analogous technique to study the behaviour of a single polymer which is “tethered to a hard wall immersed in explicit solvent and undergoing shear flow”.

All the aforementioned hybrid schemes combine a mesh-based solver with the (particle-based) molecular dynamics approach. These kinds of coupling schemes also represent the subject of the implementations and developments which are discussed in the following sections. However, it shall be noted that also purely particle-based hybrids have been successfully employed to bridge the gap between molecular and meso-/macroscopic flow descriptions. Examples for the latter comprise DSMC-DSMC couplings [23] or the triple-scale simulation by Buscalioni et al. [36] “in which the atomistic, coarse-grained, and continuum descriptions of the liquid are concurrently coupled.” Besides, Fedosov and Karniadakis recently presented works towards triple-scale simulation [49] of blood flow using molecular dynamics, dissipative particle dynamics and a Navier-Stokes solver; the respective triple-scale scheme was shown to scale on peta-flop systems [72].

Although a variety of schemes has been proposed, no common coupling software has been developed so far. To the author’s opinion, this may be due to the physical complexity of the respective couplings; a consistent description at the interface between continuum and molecular region needs to be provided. Due to the different model descriptions and the change from grid- to particle-based modelling, this represents a very difficult task. Still, many ingredients of the different hybrid schemes are very similar which strengthens both the need and the possibility to establish a common code base for the molecular–continuum coupling.

In the section 13.2, the main ingredients for coupling a grid-based continuum solver with molecular dynamics are reviewed. Afterwards, the coupling method of Dupuis et al. [43] is used to analyse the single coupling steps and determine a general interface structure for a
common coupling software. Within this context, a prototype coupling is established using the spatially adaptive Lattice Boltzmann implementation of the Peano framework and the molecular dynamics framework MarDyn [19, 20]. The respective prototype developments are described in Sec. 13.3. Based on this software analysis, the macro-micro-coupling tool (MaMiCo) is developed in Sec. 13.4 which builds a common implementation base for massively parallel hybrid molecular–continuum simulations in two and three dimensions. Major parts of the sections 13.3 and 13.4 have been published in [135, 139].

13.2 Exchange and Conservation of Flow Quantities: Models and Algorithms

For spatial or temporal coupling of molecular dynamics and macroscopic solver instances, the conservation and exchange of flow quantities between the two systems needs to be guaranteed and established. The respective quantities are given in form of the leading hydrodynamic moments, that is mass, momentum and energy. The transfer of each quantity bares particular challenges, see amongst others [95] for an extensive overview on existing approaches and Fig. 13.1 for a brief sketch of the methods. In the following, possible algorithmic approaches to exchange mass and momentum are described. As it is only isothermal setups that are to be simulated within the scope of this thesis, energy exchange between molecular and continuum solvers can be neglected. However, the conservation of energy within the molecular dynamics system still needs to be assured. The latter point is addressed at the end of this section.

First, the exchange of mass between molecular and continuum solvers shall be discussed. With molecular dynamics representing a particle-based method, it does not belong to the group of purely incompressible simulation methods. Instead, it can also capture compressibility effects on the molecular scale. Coupling molecular dynamics to an incompressible Navier-Stokes solver hence results into two major facts:

1. As the density is constant within the Navier-Stokes solver region, neither mass nor pressure need to be mapped from molecular dynamics to Navier-Stokes. This can for example be understood by considering the algorithm for solving the incompressible system from Sec. 6.2: when velocity values are written back from molecular dynamics to the Navier-Stokes solver, the pressure can be directly evaluated in the next time step.

2. As the density is not constant in every control volume within the molecular dynamics system, respective density or pressure values need to be prescribed in the molecular
dynamics region, if the macroscopic flow quantities shall be held consistent between the molecular dynamics and the Navier-Stokes region.

With Lattice Boltzmann methods working in the slightly compressible regime, a mass value is automatically associated to each Lattice Boltzmann cell, analogously to the local pressure value. From this point of view, a mapping of mass between molecular dynamics and Lattice Boltzmann sounds more convenient. Indeed, it is often compressible Navier-Stokes solvers that are coupled to molecular dynamics instead of considering the idealised incompressible case, see for example [33]. Second, as mentioned previously, thermal fluctuations can be defined for the fluid for both compressible Navier-Stokes and Lattice Boltzmann methods, rendering the coupling to molecular dynamics more consistent.

In order to map mass from the molecular to the macroscopic system, a sampling over the local number of molecules within a particular control volume needs to be established. The control volume is typically defined by the position and size of the discrete grid cell of the respective macroscopic solver. In the simplest scenario, one could imagine to determine the number of molecules within the grid cell in each MD time step, average over these numbers for several subsequent MD time steps and so obtain valid data for the macroscopic solver. It shall be noted, that this method may not work for fluids at lower densities [95]: due to strong correlations between the single time steps, this procedure may result in bad approximations for the average values. Instead, the time interval between two samples needs to be determined from the respective correlation time.

Going from the continuum to the molecular world, the transfer of mass to the molecular system resembles an insertion or removal of molecules. Here, another problem immediately becomes apparent: although mass is given in form of a smooth continuous quantity in the continuum simulation, it is discrete in the MD simulation and given by the number of molecules in each control volume. By the definition of the molecular method, a perfect match of molecular and continuum mass cannot be established since the exchange of mass on the molecular side is restricted to multiples of the mass of a single molecule. The arising error depends on the size of each control volume and the number density of the molecular system: for huge control volumes or larger number densities, the local error in the mass distributions may play a negligible role. As mass cannot “disappear” within the coupling step when going from the continuum to the molecular system, different approaches can be used to still guarantee its conservation when it is exchanged between continuum and molecular systems. Two exemplary methods shall be considered at this stage. In a first approach, the mass that cannot be introduced into the MD system may be sent back to the continuum solver. Mass is consequently conserved in each time step. However, this procedure does not only imply variations within the MD system, but also yields mass changes in the continuous system which can lead to additional errors. Another approach consists in storing the excess mass in a separate buffer over time: assume that mass needs to be transferred between the two systems many times during the hybrid simulation. For example, we might want to couple to molecular dynamics every hundred to thousand Lattice Boltzmann time steps. Then, in the first coupling to molecular dynamics, mass is sent to the MD system and stored in the buffer. It is inserted into the MD system, until only a small amount of mass—in the ideal case, this amount is smaller than one molecule mass—remains in the buffer. This mass is kept in the buffer until the next coupling from continuum to the molecular system occurs. Following this procedure, the new mass insertion can be carried out, inserting everything from the buffer up to the minimal threshold again. Hence, mass is conserved over time, but it is not conserved locally (in space and time). Depending on the underlying coupling strategy, one or the other method may be more suitable.

Next, the algorithmic realisation of mass insertion and removal within the molecular dynamics simulation shall be discussed. For both insertion and removal in rarefied fluid flow, i.e. in gas flows, a simple random procedure may be sufficient. If a molecule shall be introduced within a given volume, a random position is chosen within this volume and the molecule is inserted at this position. In case that the molecule is very close to the position of another molecule at this randomly chosen position, strong repulsive molecular forces between these molecules may occur, leading to severe restrictions of the time step or stability of the method. In this case, a new random position is sampled until a valid position is found.
Figure 13.2: Two subsequent iteration steps (upper row from left to right and lower row) of the USHER scheme [34]. The new molecule is coloured according to its potential energy within the molecular system (red resembles very strong repulsive forces whereas green denotes a state close to the desired potential energy). The linked cells that are used in the potential and force evaluations are marked in orange. The molecule is inserted at a random position and feels strong repulsive forces. The overlap is resolved in the first iteration step, however, the potential energy of the molecule is still too big. After the next potential and force evaluation, the molecule is moved into the centre of the energy hole according to the USHER update rule and reaches the desired energy level.

Similar to the random insertion, a random removal can be carried out, if mass needs to be removed from the molecular system. Here, a list is created, containing all molecules within the considered control volume. One molecule is chosen at random and removed from the simulation.

For the simulation of dense fluids such as liquids, the insertion technique from above cannot be applied anymore. Due to the very strong potential energy contributions from the densely packed molecules within each control volume, it is difficult to find a valid molecule position by purely random-based position choices. A severe improvement to particle insertions in dense systems has been proposed by Delgado-Buscaldi and Coveney [34]. It is illustrated in Fig. 13.2 and is known as the USHER scheme. Given a particular target energy $U_{\text{target}}$ at which the molecule should be inserted, the scheme starts by choosing a random initial position of a molecule. Now, the iterative USHER scheme starts: in iteration step $i$, the total force $F_i$ acting on the molecule is evaluated, as well as its initial potential energy $U_i$. If the potential energy is not close enough to $U_{\text{target}}$, that is if $\|U_i - U_{\text{target}}\|/\|U_{\text{target}}\| > \text{TOL}$ for a given tolerance TOL, the molecule is moved along the trajectory of the force vector towards the respective energy level. Hence, the molecule is moved towards a higher energy level for $U_i < U_{\text{target}}$, resembling a step along the direction $-F_i$, or towards a lower energy level for $U_i > U_{\text{target}}$, following the direction of the force $F_i$. Due to the dense packing of the molecules, the latter case is the more common one. The step size $\Delta s$ for the molecule
movement is chosen according to the following rule [34]:

\[
\Delta s = \begin{cases} \\
\Delta s_{\text{overlap}} & \text{if } U_i > U_{\text{overlap}} \\
\min \left( \Delta s_{\text{max}}, \frac{U_i - U_{\text{target}}}{F_i} \right) & \text{if } U_i \leq U_{\text{overlap}}.
\end{cases}
\] (13.10)

Hence, if the potential energy of the inserted molecule is bigger than a threshold \(U_{\text{overlap}}\) (which identifies molecular overlaps), the molecule is moved far enough to resolve the overlap conflict. Delgado-Buscalioni and Coveney suggest to choose \(\Delta s_{\text{overlap}} = 0.9\sigma - (4\epsilon/|U_i|)^{1/\sigma}\) for single-centred Lennard-Jones molecules. In case that no overlap situation occurs, the step size is chosen from the ratio of the potential energy difference and the magnitude of its derivative \(F_i := -\nabla U_i\). This is consistent with a Taylor expansion of the potential energy up to first-order terms. Using this step size criterion, the USHER search resembles standard steepest descent methods. Besides this characteristic, a maximum displacement \(\Delta s_{\text{max}}\) is defined and chosen if the steepest descent-like expression yields a very large value for the displacement \(\Delta s\). This mechanism hence suppresses too large step sizes and jumps near "energy holes".\(^{27}\) This maximum displacement has been predicted in [34] for single-centred Lennard-Jones molecules to be optimally chosen as \(\Delta s_{\text{max}} \approx 0.1n^{-1.5}\) with \(n\) representing the mean number density of the molecular system.

Although the scheme presented so far may explore the energy landscape and hence provides a much more sophisticated means to particle insertions than randomised position choices, it may still happen that the initial position for a molecule is chosen within a very densely packed region. Hence, many iteration steps of the scheme are necessary to find a valid final position. Besides, a molecule can still be trapped in bigger-sized energy holes which do not provide any location with the target energy \(U_{\text{target}}\). The molecule may consequently move around within this hole according to the local steepest descent-based optimum search, but can never reach an acceptable energy state. To resolve this issue, a restart of the USHER algorithm is triggered each time when either the number of iteration steps exceeds a certain threshold or when an increase in the molecule’s energy state is encountered, assuming a continuous decrease of the energy within the previous iteration steps.\(^{28}\)

Optimal parameter choices for the USHER scheme have been mentioned within this section for single-centred Lennard-Jones molecules. However, choices for several parameters—relative energy tolerance, the maximum number of iteration steps or the maximum number of restarts—have not been discussed so far, as these may strongly depend on the underlying molecular system and its particular molecular properties such as number density or the depth of the Lennard-Jones potential \(\epsilon\). An extension of the USHER scheme to multi-centred molecule types has been presented in [32, 102]. The algorithm requires further modifications, such as the inclusion of rotations during the particle location update. Besides the displacement \(\Delta s\), also an appropriate angle needs to be determined from the intermolecular forces and energies. With the inclusion of the rotational degrees of freedom, additional difficulties arise and even more parameters enter the USHER scheme. For example, minimum and maximum angles that are allowed during the update of a molecule’s orientation in space need to be defined. If the minimum angle is chosen too small, oscillatory movements of the molecule close to a certain configuration may be observed. If the maximum angle is chosen too big, the molecule might easily rotate over the ideal orientation, similar to translational jumps over the energy holes. Concluding, the USHER scheme provides an efficient means to particle insertions in dense liquids. However, several degrees of freedom within the scheme require particular tuning to further improve efficiency in coupled molecular–continuum scenarios.

Having discussed the insertion of molecules, it is the removal of molecules in dense molecular systems that shall be addressed next. The severe problem of overlap situations between

---

\(^{27}\)An “energy hole” is a region at very low potential energy within the molecular domain. For too large values of \(\Delta s_{\text{max}}\), the USHER search for an optimal position with respect to the potential energy may jump over an energy hole, abstaining from the exploitation of the local regions within the hole.

\(^{28}\)If the initial energy \(U_i\) is bigger than \(U_{\text{target}}\), the molecule’s energy shall continuously decrease and approach the target energy. If the energy increases again, the algorithm jumped over the optimal location. The same arguing holds for an initial energy \(U_0 < U_{\text{target}}\).
molecules may only occur in molecular insertions. Hence, the removal is expected to perform more stable. Different removal techniques have been discussed amongst others in [95] such as removal by random (randomly choose a molecule within the control volume and remove it), momentum-based removal (remove the atom whose velocity matches the required momentum transfer between molecular and continuum solver best) or energy-based removal (remove the molecule whose energy matches the required energy transfer best). In the following, considerations shall be restricted to the random removal technique.

Besides the exchange of mass between the continuum and the molecular simulation, it is also the exchange of momentum that needs to be taken into account. In order to change the momentum within the MD system, different approaches can be used. One may group these approaches into two categories: velocity and force modification methods.

**Velocity modification methods** directly modify the velocities of individual molecules. An example is described in [43]: given a velocity \( u_{\text{macro}} \) from the continuum solver that shall be imposed on the molecular system within a particular sub-region, the velocities \( v_p \) of all molecules within this sub-region are updated according to

\[
v_p := v_p + \lambda (u_{\text{macro}} - u_{\text{ave}}) \tag{13.11}
\]

with relaxation parameter \( 0 < \lambda \leq 1 \) and the current average velocity \( u_{\text{ave}} \) sampled over all molecules in the sub-region. Using this method over subsequent time steps, the average velocity is relaxed towards the target velocity \( u_{\text{macro}} \). Similar to the relaxation process, one may also additively increase the average velocity inside a volume. This method may be used, if a given momentum \( j \) shall be introduced over subsequent time steps: assume that a momentum \( j \) shall be imposed on the molecular system over \( N \) time steps. Then, each molecular velocity within the given volume may be modified as follows:

\[
v_p := v_p + \frac{1}{M \cdot N \cdot m_p} j \tag{13.12}
\]

where \( M \) denotes the current number of molecules inside the considered volume. In one time step, the momentum in the volume is thus changed by \( m_p \sum_{p=1}^{M} \frac{1}{M \cdot N \cdot m_p} j = \frac{1}{N} j \). After \( N \) time steps, the momentum \( j \) is hence transferred to the molecular system.

Instead of directly changing the individual molecular velocities, the force modification methods rely on imposing additional forces to account for the respective momentum modifications. The required forces can be obtained from their relation to the momentum flux. In the following, the considerations are restricted to velocity modification methods. Details on force modification methods can be found in various articles on coupled molecular–continuum schemes by Delgado-Buscalioni or the PhD thesis of Kalweit [95].

Finally, energy transport and respective conservation properties require particular considerations in coupled simulations. Since only isothermal scenarios are described in the following, the energy of the molecular system must not be changed by the exchange of mass and momentum between the macroscopic and the molecular solver. This implies on the one hand that temperature needs to be constant within the molecular domain and that the potential energy has to be conserved on the other hand. Considering the conservation of temperature, a standard approach to achieve this requirement is the application of thermostats; see standard literature on MD simulations for details on different thermostat mechanisms [71, 148]. Thermostats can be used to regulate the temperature within the MD system and, hence, push the temperature towards a prescribed target temperature in each MD time step. In the following, a simple Berendsen thermostat [11] is used: given an average velocity \( u_{\text{ave}} \) of the MD system and a current temperature value \( T \), a new temperature \( T_{\text{target}} \) can be imposed to the molecules by modifying their velocities as follows:

\[
v_p := u_{\text{ave}} + \sqrt{\frac{T_{\text{target}}}{T}} (v_p - u_{\text{ave}}) \tag{13.13}
\]

This mechanism rescales the molecular deviations from the average velocity \( u_{\text{ave}} \) and hence sets the correct temperature in the molecular system. Conserving the potential energy within the molecular dynamics simulation is more complicated. The potential energy may change
when inserting or deleting molecules from the molecular simulation. As discussed previously within this section, the USHER scheme takes into account the local potential energy of the molecules. This guarantees that newly inserted molecules are inserted at a satisfying energy level, close to the mean potential energy level. Similarly, using the random removal technique for deletion of mass, a molecule is in average deleted at the correct mean potential energy. However, with the potential energy arising from the sum over the contributions of all $N$ molecules of the molecular system, the insertion/deletion of a single atom yields a change in the potential energy of $1 + 1/N$ or $1 - 1/N$, respectively. For most simulations close to the incompressible limit, i.e. for simulations where the number of molecules within a fixed volume is expected to stay approximately constant over time, the number of deletions and insertions is expected to be of same order. Therefore, in the following scenarios of interest, the conservation of potential energy due to mass exchange is expected to be globally fulfilled, up to natural fluctuations that arise from the local exchange mechanisms.

Besides the mass exchange, it is also the lack in periodicity across the global boundaries which affects the potential energy landscape. Most molecular dynamics simulations are executed within a box volume, using periodic boundary conditions at the outer boundaries of the simulation box as illustrated in Fig. 13.3 on the left. Based on the periodicity, infinite systems may be emulated: the molecules close to a boundary “feel” the energy and force of the molecules which are located at the opposite boundary. However, in most coupling scenarios, periodic conditions may not represent a sufficient method of choice, due to a lack of periodicity in most flow problems of interest. As a consequence, considering the near-boundary region in Fig. 13.3 on the right, no molecule populations are predefined in the outer boundary layer which is illustrated by the red cells. The missing energy and force contributions thus need to be emulated to yield physically correct energy values for the near-boundary molecules and also retain the correct pressure inside the molecular system. Revising the requirements to the boundary forces from Werder et al. [180], the respective boundary model needs to exert the correct pressure and minimise local disturbances close to the boundary. Several boundary models have been developed and successfully applied in coupled molecular–continuum simulations [33, 55, 101, 142, 180]; for a comparison of various force models, see [180]. One of the most promising approaches which takes into account the atomic structure of the underlying fluid is reported and validated by Werder et al. for supercritical fluids. It uses the radial distribution function $g(r)$, that is the probability to find an atom close to the current one in a given distance $r$, to compute the average force contribution from the boundary zone, cf. Fig. 13.4. The grey-coloured region denotes the volume within the continuum domain $\Omega_C$; this volume shall be denoted with $\Omega_B$. It potentially carries additional force contributions that act on the current molecule. If the
radial distribution function $g(r)$ is known, the average boundary force $F_B(x)$ from $\Omega_B$ onto the particle at position $x$ can be computed via integration over this volume:

$$F_B(x) = n \int_{y \in \Omega_B} g(\|y - x\|)F(y - x)dy$$  \quad (13.14)

where $n$ is the mean number density and $F$ the interaction force between two molecules. Werder et al. measure the radial distribution function from a periodic molecular dynamics simulation, perform a polynomial fit to the arising function and solve the integral expression from Eq. (13.14) analytically. This analytical derivation yields a particularly efficient force method since only one single additional polynomial evaluation needs to be performed for each molecule near the computational boundary. However, for other than plane boundaries, e.g. boundaries that are represented by the corners of the box-like molecular dynamics domain, an analytical derivation of the overall boundary force is not possible anymore. In order to allow for arbitrary boundary shapes, a quadrature approach is required. In the following, the midpoint rule is used to evaluate the integral. First, the linked cells which contain $\Omega_B$ for the current atom are determined. Second, their volume is discretised using a set of $M_D$ fine cells per linked cell. The integral is then approximated via evaluating the force $F(y - x)$ and the radial distribution function $g(\|y - x\|)$ in the midpoints $y$ of all fine cells, resembling a zero-order approximation of the respective functions on the fine cell volume. The results of the single fine cells are finally summed up and weighted by the fine cell volume. Depending on the refinement level that is applied in the further discretisation of each linked cell, a better approximation of the mean boundary force can be achieved. However, each fine cell evaluation corresponds to an additional intermolecular force evaluation and yields additional computational costs: assume a three-dimensional molecular dynamics simulation, using a cut-off radius $r_c = 2.4\sigma$ and a number density $n = 0.6$. Then, a linked cell contains $0.6 \cdot 2.4^3 \approx 8$ molecules. If the radial distribution function-based boundary force method shall come at similar cost as periodic boundaries, a total number of 8 cells can be used for the discretisation of a linked cell, implying a $2 \times 2 \times 2$ fine cell discretisation. Now, assume a molecule to be exactly placed on the open boundary. Due to the very coarse discretisation, the regions of the radial distribution function which show a high slope, cf. Fig. 13.5, may not be captured sufficiently when evaluating the boundary force for this molecule. The force that acts on the molecule is therefore expected to be underestimated. A finer discretisation would thus be necessary in this case to yield a better approximation of the boundary force. A first step to reduce computational cost and to improve the integral approximation is to only consider those regions where the radial distribution function takes non-zero values.
Studies on the impact of the fine cell discretisation on a molecular system can be found in Sec. 17.2.

### 13.3 Prototype Implementations: Identification of Software Requirements

The descriptions from the previous section and from Fig. 13.1 demonstrate that several components are required to establish a hybrid molecular–continuum simulation: mechanisms for mass, momentum and energy transfer, open boundary treatment in the MD simulation, sampling strategies to obtain averaged quantities for the continuum simulation and mappings of the exchanged quantities between the two solvers. Until now, the algorithmic realisation of the transfer mechanisms was highlighted using different examples and explicit algorithms such as the default USHER scheme or the thermostats in isothermal simulations. Most of these schemes are presented in various papers on hybrid molecular–continuum papers. However, only little information on efficient and modular implementation concepts of the respective coupled simulations has been provided in literature yet. This represents a big issue: to the author’s knowledge, no general coupling software for molecular–continuum simulations is available yet. As a consequence, the testing and verification of new coupling approaches may result in additional implementational overheads during the code development since all coupling steps either need to be re-implemented or re-arranged. Due to the complexity and sensitivity of hybrid scenarios, the verification may become a very time-consuming and non-trivial task.

For these reasons, a general, modularised coupling software for hybrid molecular–continuum simulations may yield a high level of software modularity and also ease the process of hybrid simulation developments. In order to determine the software requirements and to formulate and identify interfaces for the different coupling components from the previous section, a hybrid molecular–continuum prototype has been developed. The prototype is based on the state-based coupling approach from Dupuis et al. [43] which is considered to comprise all common coupling steps. It hence builds a valid basis for coupling software developments. The prototype coupling is established using the spatially adaptive Peano-based Lattice Boltzmann implementation (see Chap. 10) and the molecular dynamics framework MarDyn [19] which has previously been used for different chemical engineering applications.

Before stepping into the software analysis, the coupling scheme is shortly reviewed. It is illustrated in Fig. 13.6. The molecular dynamics region is embedded into the Lattice Boltzmann domain. Overlap regions are defined in which the physical quantities are transferred from Lattice Boltzmann to molecular dynamics and vice versa. The respective overlap regions typically comprise two or more Lattice Boltzmann cells, providing a smooth transition.
between the flow descriptions. Going from the molecular dynamics to the Lattice Boltzmann simulation, average velocities are extracted from the MD system and imposed onto the particle distribution functions of the LB system by an additional acceleration term. This term reads

$$a = \frac{1}{dt} \left( u^{MD}(x, t + dt) - u^{LB}(x, t + dt) \right)$$ (13.15)

where $u^{MD}$ denotes the imposed velocity value from the MD simulation and $u^{LB}$ the velocity of the LB simulation when no acceleration term is considered. Going from the Lattice Boltzmann to the molecular dynamics simulation, the outer overlap region of the MD domain is used to relax the average MD velocities of each control volume, i.e. each LB cell, towards the provided velocity from the Lattice Boltzmann simulation; the relaxation process is described by Eq. (13.11). Assuming a channel flow-like scenario, the molecules are in average expected to move from left to right. The USHER and the random particle removal algorithm are used to insert/remove molecules that leave the MD domain on the left/right side on the opposite side of the simulation domain. Reflecting walls are used at the other boundaries to keep the molecules inside the domain. Besides, the RDF-based boundary force formulation from Eq. (13.14) is used to model the interparticle potential at the boundaries. In order to retain the temperature level, a thermostat is applied in the boundary cells of the MD domain. Since the coupling combines a two-dimensional Lattice Boltzmann simulation with a three-dimensional molecular dynamics simulation, the MD simulation is assumed to be periodic in z-direction. The thermostat is applied to the molecular velocity components in that direction.

In the coupling described by Dupuis et al., only momentum is exchanged between the molecular and the Lattice Boltzmann solver. In order to map the momentum in terms of averaged velocity values from molecular dynamics to Lattice Boltzmann, several steps need to be taken. First, as pointed out in Sec. 13.2, a control volume—corresponding to one Lattice Boltzmann cell in this case—needs to be defined. Second, a spatial and temporal averaging over all molecules within this volume needs to be carried out. The coupling software hence needs to provide both description of the desired control volume and an interface which allows to iterate over all molecules within the volume. Third, besides iterating, the access to the physical quantities of each molecule is required, that is the coupling software needs to have access to the molecule’s properties, such as the molecular velocity in the present example. It
may further be remarked that a re-scaling of the physical quantities might be necessary when
going from molecular dynamics to Lattice Boltzmann, since different dimensionless formulations
may be available on both solver sides. Fourth, the transfer of the sampled averaged
quantities from the MD solver to the Lattice Boltzmann solver needs to be established. In a
more general scenario, this transfer may not only include the velocity or momentum within
the control volume, but may also comprise mass and energy values. Moreover, depending on
the type of coupling (flux- vs. state-based coupling), the coupling scheme dictates whether
to transfer state or flux variables. From a technical point of view, however, this does not
impose any significant differences with respect to the transfer of the flow quantities. Finally,
the quantities that are transferred to the Lattice Boltzmann simulation need to be imposed
to the mesoscopic flow system. In the present example, the averaged velocities from the
MD simulation are incorporated on LB side via an additional forcing term which pushes the
Lattice Boltzmann cell values towards the correct momentum. However, the imposition of
the physical quantities depends on a variety of prerequisites, such as the type of continuum
solver (Lattice Boltzmann, Navier-Stokes, etc.) and its respective degrees of freedom or the
type of coupling (state-coupling, flux-coupling). From the author’s point of view, a generic
interface definition that obeys all these points is impossible due to this extremely low degree
of commonality.

Next, consider the transfer from Lattice Boltzmann to molecular dynamics. For the coupling
scheme from Dupuis et al., the USHER scheme is applied in the left and right boundary
region of the overall MD domain. Besides the read- and write-access to the quantities of a
single molecule, this scheme needs to evaluate both potential energy and force of the newly
inserted molecule. A coupling software hence needs to comprise an interface that iterates
over all molecules in the nearby-region of the new molecules, that is over the corresponding
3D linked cells, and computes the respective force and energy contributions. The USHER
scheme further requires the evaluation of the mean potential energy in the region of interest
to determine the target energy level at which the new molecule shall be inserted. The impos-
sition of reflecting walls in the other boundary regions represents an additional coupling step
in the coupling scheme. Similar to the USHER-based insertion for the main flow direction,
it requires access to the molecules that left the computational domain of the MD simulation.
If the molecules are available, their molecular velocities can be reversed and the insertion
can be carried out. For both USHER- and reflecting wall-based insertion, additional inter-
faces need to be implemented by the MD simulation which yield access to its main data
structures: the newly inserted molecules need to be added to the molecule data structure as
well as to the linked cell structure.

The velocity transfer via the velocity relaxation scheme from above requires an iteration
mechanism which loops over all molecules: in a first iteration over the respective control
volume, the average velocity in this region is evaluated. In the second iteration, the molecu-
lar velocities are relaxed towards the target velocity from the Lattice Boltzmann simulation.
The application of the thermostat to the molecular dynamics simulation can be established
analogously to the velocity relaxation and hence requires analogous interface definitions and
implementations. It may further be remarked that the same formalism applies to force modi-
fication methods as well.

As a last step in the coupling, the RDF-based boundary force needs to be applied to the
molecules that are close to the outer MD boundary. In a first sight, this task sounds
trivial in terms of defining and implementing the respective interfaces: a generic method
addBoundaryForce(Molecule &, const Vector& mdDomainSize, const Vector& mdDo-
mainOffset) can be defined in which a molecule obtains additional force contributions if
it is located close to the boundary of the MD domain. These contributions can then be
evaluated using the RDF-based force formulation or any other force definition. The imple-
mentation is less trivial since other molecular quantities may enter the force evaluation and
need to be extracted from the MD simulation on-the-fly. As an example, consider the RDF-
based force from Eq. (13.14): one quantity entering the force computation is the number
density. For close-to-incompressible systems, the density may be assumed to be constant.
For compressible systems, however, the density can strongly vary inside the MD region and
hence needs to be locally taken into account. As a consequence, the respective interface
Figure 13.7: Velocity field around a sub-grid-sized carbon-nanotube in a hybrid molecular dynamics–Lattice Boltzmann simulation [135]. The latter applies a three-level spatially adaptive grid and embeds the molecular dynamics simulation on the finest grid level, respectively. The molecular dynamics region is surrounded by the black thick line. The nanotube is located in the centre of the blue-coloured low- to zero-speed region. The nanotube is not shown in the grid-based plot, nor is it explicitly visible to the Lattice Boltzmann geometry.

The interfaces that have been identified during the prototype implementation are listed in Tab. 13.1. Based on the discussion from above, MarDyn and the Peano-based Lattice Boltzmann application were coupled and applied to different channel flow scenarios. Further details can be found in [85]. One of the scenarios is depicted in Fig. 13.7: a three-level spatially adaptive Lattice Boltzmann simulation is coupled to MarDyn to simulate the flow around a small carbon-nanotube.

13.4 The Macro-Micro-Coupling Tool (MaMiCo)

In most hybrid molecular–continuum schemes, overall concepts such as averaging procedures or particle insertions are very similar (particular examples were presented in Sec. 13.3). A typical sequence of algorithmic phases within a coupling cycle of a molecular–continuum simulation reads:

1. Run macroscopic solver and impose quantities from molecular dynamics simulation
2. Extract quantities from macroscopic solver and transfer them to the molecular dynamics simulation
3. Run molecular dynamics solver and impose quantities from macroscopic simulation
4. Extract quantities from molecular dynamics (averaging) and transfer them to the macroscopic solver

Hence, besides extraction and imposition of flow quantities, it is the transfer of the quantities between the two solvers that needs to be carried out and typically can be accomplished exactly the same way for most coupling schemes. A common implementation of both transfer methodology and quantity imposition therefore becomes very desirable when developing and testing different hybrid schemes.

Another important point arises from the fact that the computational time required by the molecular dynamics simulation in most scenarios of interest is orders of magnitude bigger
# Functionality | Interfaces/ Requirements | MaMiCo implementation
---|---|---
**Sampling/ Averaging** | Definition of control volume | MacroscopicCell
Molecule iterator | MoleculeIterator
Molecule proxy | MoleculeWrapper
**Quantity extraction and transfer** | Rescaling of quantities | TransferStrategy
Quantity exchange | MacroscopicCellService, MDSolverInterface
**USHER scheme** | Molecule proxy | MoleculeWrapper
Local force and energy evaluation | MDSolverInterface
Mean potential energy evaluation | MDSolverInterface
Access to molecule data structure | MDSolverInterface
Access to linked cell data structure | MDSolverInterface
**Reflecting wall boundaries** | Molecule proxy | MoleculeWrapper
Access to molecule data structure | MDSolverInterface
Access to linked cell data structure | MDSolverInterface
Access to molecules that left the computational domain | MDSolverInterface
**Velocity relaxation** | Molecule iterator | MoleculeIterator
Molecule proxy | MoleculeWrapper
**Thermostat** | Molecule iterator | MoleculeIterator
Molecule proxy | MoleculeWrapper
**Boundary forcing** | Molecule proxy | MoleculeWrapper
Molecule iterator | MoleculeIterator
Forcing type-dependent interfaces | |

Table 13.1: Identified interfaces and requirements for the coupling software. The required functionality and respective interfaces are listed in the first and second column. The counterparts to the identified interfaces in the macro-micro-coupling tool (MaMiCo) are listed in the third column.

than the time needed by the continuum solver. For example, the computational times have been studied by Barsky et al. [10] in the case of molecular–continuum simulations of polymers immersed in shear flows. In their scenarios, the computational time of the continuum solver only amounts to 0.01% of a single MD time step. In order to speed up the computations, a (massively) parallel molecular dynamics solver may therefore become necessary, as well as a parallel implementation of the coupling mechanisms.

In view of the considerations from above and the discussion from Sec. 13.3, the macro-micro-coupling tool (MaMiCo) has been developed [135, 139] within the scope of this thesis. It is meant to facilitate the development of massively parallel hybrid molecular–continuum implementations in 2D and 3D and is written in C++. The major steps in the coupling algorithms—extraction, transfer and imposition of physical quantities—are encapsulated in separate modules and are executed on **macroscopic cells**, cf. Fig. 13.8: quadratic (2D) or cubic (3D) cells are introduced, covering the linked cell structure of the molecular dynamics simulation. The transfer of the flow quantities is established between the macroscopic solver and MaMiCo’s cell structure or the molecular dynamics simulation and MaMiCo’s cell.
structure, respectively. In case of parallel computations, the macroscopic cells of MaMiCo are located on the same process as their embedded linked cells. With molecular dynamics representing the computationally most intensive part, this choice has been taken to avoid additional communication and synchronisation costs between the processes during the quantity transfer to/from the molecular system.

The modular design and functionality of the coupling tool is depicted in Fig. 13.9 and is discussed in the following. Coupling a continuum solver to MaMiCo, two methods need to be provided by the programmer in the `MacroscopicSolverInterface`, deciding whether a particular grid cell at position `position` sends/receives information to/from MaMiCo:

- `bool receiveMacroscopicQuantityFromMDSolver(position, ...)` for receiving and
- `bool sendMacroscopicQuantityToMDSolver(position, ...)` for sending data.

Based on this information, the coupling tool sets up the macroscopic cell structure and the mapping relations between the macroscopic solver and the macroscopic cells from MaMiCo. In the parallel case, the coupling tool collects the information from all ranks, determines the counterparts to the grid cell positions of the continuum solver in terms of MaMiCo’s cells and sets up the communication topology between the involved ranks. On the MD side, the latter steps can be skipped since the cells of MaMiCo are expected to be located on the same processes as the embedded MD simulation domain. When transferring information from the continuum to the molecular solver, the continuum solver may simply extract values for mass and momentum descriptions, store them in buffers `mass` and `momentum` and call `sendMacroscopicQuantityToMDSolver(position, mass, momentum)` of the `CouplingMacroscopicSolverService` (not shown in Fig. 13.9) in each grid cell. Based on the decision of `sendMacroscopicQuantityToMDSolver(...)` of the macroscopic solver’s interface, these quantities will then be sent to MaMiCo. An analogous procedure can be used to receive data from MaMiCo on the continuum side, using the method `receiveMacroscopicQuantityFromMDSolver(position, mass, momentum)`.

The `TransferStrategy` defines an interface which allows for the interpretation and conversion of state- and flux-based quantities. With MaMiCo’s core functionalities operating in a state-like manner, the macroscopic cells of MaMiCo hold buffers for mass and momentum contributions from the continuum or the molecular simulation. Now, assume for example, that the macroscopic solver can only extract the absolute value of mass within its grid cells and sends this quantity to MaMiCo. Then, it is not this total amount of mass that needs to be introduced at the MD domain boundaries, but only the difference between the macroscopic solver mass and the molecular mass within the respective MaMiCo cell. Implementing the respective conversion method of the `TransferStrategy` exactly allows for
this kind of operations on the mass and momentum data\textsuperscript{29}. Besides, the TransferStrategy can further be used to re-scale the transferred quantities according to the dimensionless forms that are used for both molecular and continuum solver. Similar conversions may arise before sending molecular data to the macroscopic solver and can also be implemented via the TransferStrategy; converting the values immediately via the TransferStrategy also facilitates the process of receiving the data on the continuum side. After calling receiveMacroscopicQuantityFromMDSolver(position, mass, momentum), the values in the mass and momentum buffers are already converted so that the stored values can directly be imposed to the continuum solver. No further conversion or adoption on the continuum side is necessary.

Having discussed the steps to couple MaMiCo and the continuum solver, the coupling to the molecular dynamics simulation shall be addressed next. Three different interfaces need to be provided by the molecular dynamics solver. A MoleculeWrapper yields access to the molecular quantities such as momentum or the force that acts on the molecule. Besides, a MoleculeIterator is required which traverses all molecules within a given linked cell. Last, the MDSolverInterface gives access to global parameters and functionality of the MD solver. These global settings comprise amongst others the methods addMoleculeToMDSimulation(molecule) and deleteMoleculeFromMDSimulation(molecule) which are needed by MaMiCo to add/ remove molecules to/ from the MD simulation, or calculateForceAndEnergy(molecule, linkedCellIndex) which computes the forces and energies that act onto a newly inserted molecule and stem from interactions with molecules of a particular linked cell.

Similar to the macroscopic solver, mass and momentum need to be inserted or extracted from the MD simulation. The insertion can be accomplished using the ParticleInsertion or the MomentumInsertion modules. The user can either use one of the existing implementations or provide an own implementation for these algorithmic steps. Currently, an USHER implementation is available for particle insertion, as well as velocity relaxation and additive momentum transfer implementations for momentum insertion, cf. Sec. 13.2. As an

\textsuperscript{29}The implementation may be established within processInnerMacroscopicCellAfterReceivingMacroscopicSolverData(MacroscopicCellA, index) of the TransferStrategy.
example, the interface definition for the momentum insertion is given below:

```cpp
template<class Molecule,class LinkedCell, unsigned int dim>
class moleculardynamics::coupling::MomentumInsertion {
    public:
        MomentumInsertion(){
            virtual ~MomentumInsertion();
        }
        /** accesses the momentum buffer of the macroscopic cell 
         * and inserts momentum at time step t. The total number of 
         * time steps for the insertion is given by numberOfMDTimesteps. 
         */
        virtual void insertMomentum(
            moleculardynamics::coupling::MacroscopicCell<LinkedCell,dim>& cell, 
            const unsigned int& t, 
            const unsigned int& numberOfMDTimesteps 
        ) const = 0;
        /** returns the number of MD time steps between subsequent momentum 
         * insertions. 
         */
        virtual unsigned int getTimeIntervalPerMomentumInsertion() const = 0;
};
```

In massively parallel simulations, the momentum of the molecular system can be modified locally. No additional communication is required in this case. This, however, does not hold for the transfer of mass, that is for particle insertions. The arising problem is sketched in Fig. 13.10: if new molecules are inserted in neighbouring linked cells at the same time, molecular overlaps can occur which result in very strong repulsive forces between the molecules and hence lead to instabilities of the coupled simulation. A strategy to resolve this problem has been presented in [31] and has been further extended in the current USHER implementation [139]. It is shown in Fig. 13.11: the domain of each process is split into $2^D$ blocks. Each block obtains a particular colour (“block-colouring”). New particles can now be introduced on different processes as long as these insertions are performed inside blocks of the same colour. After handling all blocks of one colour on all processes, the newly inserted particles are synchronised between the neighbouring processes and particle insertions inside the blocks of the next colour can be carried out. In order to further speed up the insertion process, the evaluation of the mean potential energy is carried out by an analogous technique. Therefore, the macroscopic cells inside each block are also coloured (“cell-colouring”) using $2^D$ colours. As the potential energy inside a macroscopic cell is only affected by the molecules of the cell itself and its neighbouring cells, an insertion of particles in macroscopic cells of same colour can be carried out simultaneously. After the particle insertions in all macroscopic cells of same colour within one block are finished, the mean potential energy in the other macroscopic cells is updated, and the particle insertions in the cells of the next colour are processed. The cell-colouring approach allows to minimise the number of mean potential energy evaluations. Besides, it also builds the base for shared-memory parallelisation of the USHER scheme. Combining the block- and cell-colouring hence represents a general receipt for parallel USHER-based particle insertions in simulations that apply distributed-memory, shared-memory or hybrid parallelisation concepts.

When a macroscopic quantity is imposed on the molecular system, other macroscopic quantities can be affected by the respective molecular modifications as well. For example, when introducing a new molecule via the USHER scheme, the overall momentum in the cell changes since the new molecule also moves at a certain velocity. Therefore, controllers are used to locally retain the state of the molecular system during the imposition of one particular quantity. Although a global conservation may be highly desirable, arising additional commu-
Figure 13.10: Parallel particle insertion on two neighbouring processes. Each process holds a domain of $3 \times 3$ linked cells. If particles (coloured in red) are inserted in neighbouring linked cells that are located on different processes, unphysical overlap situations can occur.

Figure 13.11: “Block-” and “cell-colouring” approach for parallel particle insertions. Each block is sub-divided into $2D$ blocks which are surrounded by coloured lines. In a distributed memory setup, the particle insertion can be executed simultaneously in all blocks of neighbouring domains. Within each block, all macroscopic cells are also coloured using $2D$ colours. The mean potential energy evaluation as well as the particle insertion can now be carried out in parallel on all macroscopic cells of same colour within each block.

Communications in case of parallel simulations may become expensive. Besides, as the macroscopic quantities of mass, momentum and energy are locally defined for each macroscopic cell, a conservation of these quantities within each macroscopic cell is reasonable; a validation of this mechanism is provided in Sec. 17.1. Staying at the example of particle insertions, a control mechanism to keep momentum and temperature constant during the insertion reads:

1. KineticEnergyController: store the current temperature $T^{old}$
2. MomentumController: store current momentum $j^{old}$
3. ParticleInsertion: insert particle into macroscopic cell
4. MomentumController: set momentum to $j^{old}$
5. KineticEnergyController: set temperature to $T^{old}$
The insertion algorithms—including the respective control mechanism—can be called from within the MD simulation. The method `distributeMacroscopicQuantities(t)` of the `MacroscopicCellService` triggers the underlying imposition schemes in each time step $t$. 

The extraction of macroscopic quantities from the molecular system, that is averaging over the atomistic data, can be incorporated via a simple callback mechanism: the MD simulation calls `processInnerMacroscopicCellAfterMDTimestep()` of the `MacroscopicCellService` which loops over all local MaMiCo cells. For each cell, the method `processInnerMacroscopicCellAfterMDTimestep(cell,index)` of the current `TransferStrategy` is triggered. A spatial and temporal averaging over the MaMiCo cell data can hence be simply implemented via the `TransferStrategy`.

Closing the description of the coupling tool, it should be remarked that boundary forces which are required in case of open boundary molecular dynamics simulations have not been incorporated into the tool yet. As mentioned earlier, boundary forces are still subject to current development. The definition of a stable interface for different boundary force types is therefore not possible yet. Several questions have to be addressed: how much “freedom” shall the interface have in terms of access to the molecular system? How much information shall be re-used from the MD simulation? For example, most MD simulations provide their own output format for radial distribution function measurements. Should those formats and respective files be used as input for the RDF-based boundary force model in the coupling tool? Still, basic implementations of the interface descriptions from Sec. 13.3 are to be realised in near future.

Besides, different kinds of parallel molecular–continuum implementations are imaginable, for example combining

- one continuum and one MD simulation. This corresponds to the case of the prototype implementation from Sec. 13.3.
- one continuum and several MD simulations. Using more MD simulations yields statistical uncorrelated molecular data and hence represents an alternative approach to spatial and temporal sampling.

In the following, the one-continuum-to-one-MD coupling approach is considered.
Part IV

Applications and Numerical Results

In the following, the algorithms and implementations developed in Part III are used for simulations of relevant scenarios from computational fluid dynamics. Each section provides respective numerical results as well as a short conclusion. The Lattice Boltzmann implementation that has been developed and explained in Chap. 10 is validated in Chap. 14. It is shown that the adaptive LB scheme in Peano shows good agreement with different benchmark data and has a performance which is comparable to well-established community codes. An analysis of the memory requirements further points out the low memory footprint of the LB application in Peano. Lattice Boltzmann simulations in the finite Knudsen range are subject of Chap. 15. Here, different channel flow scenarios are described, and the developed LB code extension that is incorporated into Peano’s LB application is applied to flow problems in microreactor geometries. The LB application thus shows its capabilities at successfully simulating flows in the slip and transition regime, approaching the molecular regime. A second example for flow simulations close to the molecular scale is given by particle transport in nanopore structures and is investigated in Chap. 16. A dynamically adaptive LB scheme incorporating multi-level collision model representations is developed in Sec. 16.1. The new dynamic mesh refinement technique from Sec. 10.5 is validated and subsequently used to simulate the short-time behaviour of a particle which is released in an oscillating flow in a nanopore. Due to the multi-level collision representations, a coarsening and cut-off approach for fluctuating hydrodynamics is applied to further speed up and simplify the simulation. In order to simulate longer time intervals, a hybrid Lattice Boltzmann–Navier-Stokes approach is used to predict the behaviour of the particle; the respective scenario is described in Sec. 16.2. For this purpose, the new optimisation-based coupling scheme for hybrid LB–NS simulations (cf. Sec. 12.2) is used. The scheme is also validated in this section, and second-order accuracy for the optimisation-based LB boundary treatment is shown. The part closes with Chap. 17 on aspects of coupling molecular dynamics and Lattice Boltzmann simulations. Various test cases and performance results for the coupling tool (cf. Sec. 13.4) are presented in Sec. 17.1. The efficiency and applicability of the tool in both sequential and massively parallel simulations are pointed out. For the latter, special focus is also put on the parallel USHER-based particle insertion scheme that has been developed in Sec. 13.4. A discussion of RDF-based boundary forcing terms is provided in Sec. 17.2. Finally, results for fully three-dimensional coupled molecular–continuum flows in channel-like geometries are presented in Sec. 17.3.
Table 14.1: Primary vortex centre in two-dimensional spatially adaptive cavity scenarios for different Reynolds numbers using the BGK collision model. The refinement is triggered near the computational boundaries. The coarsest and finest mesh sizes are given in the second and third column. The fourth column shows the results using the spatially adaptive LB solver of Peano. The reference data from Ghia et al. [65] are provided in the fifth column. The block size for the LB blocks is set to $6^D$. Due to Peano’s tripartitioning, the grid resolution of level $l$ evolves at $dx^l = \frac{1}{6^l}$. 

### 14 Lattice Boltzmann Simulations within Peano: Validation and Performance

In the following, validational results are presented for the Lattice Boltzmann implementation within the Peano framework\(^1\). Different benchmark scenarios are solved using the (adaptive) LB solver and compared to results of other research groups. Most of the validational results have previously been published in [137].

First, the lid-driven cavity scenario (cf. Sec. 5.5) is simulated. The lid of the closed box is moving at a constant velocity $u_{wall} > 0$, yielding a Reynolds number $Re = \frac{u_{wall} L}{\nu}$ where $L$ is the size of the box and $\nu$ the kinematic fluid viscosity. Due to the moving wall at the top of the box, a primary vortex is created in the centre of the box. For higher Reynolds numbers, secondary vortices can be observed in the lower corners and the upper left corner of the box. After reaching the steady state, the centre of the primary vortex is determined and compared to the reference values reported by Ghia et al. [65]. The measured vortex positions, extracted from adaptive BGK-based LB simulations, are depicted in Tab. 14.1 and show good agreement with the reference values. Similar results are obtained for MRT-based cavity simulations. For both, BGK- and MRT-cavity simulations, the velocity profiles at the cross-sections $x = 0.5$ and $y = 0.5$ for both velocity components $u$ and $v$ are measured. The respective plots for the MRT-based simulations are shown in Fig. 14.1. Except for the $Re = 3200$ scenario, where the left peak in the velocity component $v$ is underestimated, the measured profiles agree well with the reference values from Ghia et al. The characteristics of the cavity flows are hence captured correctly.

In a second scenario, two-dimensional channel flows are considered. The geometry consists of an inlet and a channel outlet that are located on the left and on the right of the rectangular domain, respectively. The upper and lower boundaries are modelled as no-slip walls. At the inlet, a parabolic velocity profile is defined whereas a fixed pressure value is prescribed at the outlet of the channel, using the boundary conditions from Sec. 5.5. Hence, the arising steady flow needs to show the well-known linear decrease in the pressure along the whole channel length and the same parabolic velocity profile at each cross-section of the channel. The scenario has been simulated at different Reynolds numbers; the flow characteristics could be captured correctly assuming a sufficiently fine resolution of the grid\(^2\).

To further validate the Lattice Boltzmann application, different benchmark scenarios from

\(^1\) Besides validation, code verification and testing has been achieved via the unit test functionality of Peano. Moreover, continuous integration including automatic testing has been applied during software development.

\(^2\) Due to the Chapman-Enskog theory, a sufficiently fine resolution is required to enter the respective asymptotics of the series expansion.
Figure 14.1: Profiles for the horizontal and vertical velocity components $u$ and $v$ in a cavity scenario, measured over a vertical and horizontal centre line at $x = 0.5$ and $y = 0.5$, respectively. The profiles are obtained from MRT-based simulations for adaptive grid resolutions $1/18-1/162$ (green line) and $1/54-1/486$ (yellow line). The triangles represent the reference data from Ghia et al. [65]. Similar results are obtained from BGK-based LB simulations.

[174] have been simulated by means of adaptive and non-adaptive LB simulations. Here, a cylinder is placed inside the computational domain. The characteristics of the flow around this obstacle are determined from the simulation data. The results for the 2D1-, 2D2- and the 3D1Q-benchmark are listed in Tab. 14.2. The evaluation is restricted to the quantities that are of major interest for the following particle-in-channel scenarios, that is the pressure drop, the drag exerted onto the obstacle and the Strouhal number. Measuring the pressure drop provides a further validation mechanism—besides checking velocity profiles and the linear decrease of pressure in Poiseuille channel flows—for the correct behaviour of the hydrodynamic pressure in the overall system. Further analysis of the pressure for flows in the transition and slip flow regime, that is for rarefied gas flows, is provided in Chap. 15. The drag coefficient defines the major characteristic value in particulate flow scenarios, since the drag force determines the translational movement of the suspended particles. The Strouhal number represents a measure for the time-dependent flow behaviour. It is a dimensionless characteristic number for the vortex separation and, hence, for the correct response of the
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Mach number</th>
<th>Coarsest resolution</th>
<th>Finest resolution</th>
<th>$c_D$</th>
<th>$\Delta P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D1, Re=20</td>
<td>0.0473</td>
<td>0.41/30</td>
<td>0.41/30</td>
<td>5.62</td>
<td>0.114</td>
</tr>
<tr>
<td>2D1, Re=20</td>
<td>0.0473</td>
<td>0.41/90</td>
<td>0.41/90</td>
<td>5.40</td>
<td>0.110</td>
</tr>
<tr>
<td>2D1, Re=20</td>
<td>0.0473</td>
<td>0.41/270</td>
<td>0.41/270</td>
<td>5.28</td>
<td>0.109</td>
</tr>
<tr>
<td>2D1, Re=20</td>
<td>0.0473</td>
<td>0.41/30</td>
<td>0.41/90</td>
<td>5.42</td>
<td>0.111</td>
</tr>
<tr>
<td>2D1, Re=20</td>
<td>0.0473</td>
<td>0.41/30</td>
<td>0.41/270</td>
<td>5.34</td>
<td>0.110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Mach number</th>
<th>Coarsest resolution</th>
<th>Finest resolution</th>
<th>$c_D$</th>
<th>$\Delta P$</th>
<th>Strouhal number</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D2, Re=100</td>
<td>0.0947</td>
<td>0.41/90</td>
<td>0.41/90</td>
<td>3.22</td>
<td>2.44</td>
<td>0.286</td>
</tr>
<tr>
<td>2D2, Re=100</td>
<td>0.0947</td>
<td>0.41/270</td>
<td>0.41/270</td>
<td>3.10</td>
<td>2.35</td>
<td>0.290</td>
</tr>
<tr>
<td>2D2, Re=100</td>
<td>0.0947</td>
<td>0.41/30</td>
<td>0.41/90</td>
<td>3.17</td>
<td>2.36</td>
<td>0.288</td>
</tr>
<tr>
<td>2D2, Re=100</td>
<td>0.0947</td>
<td>0.41/30</td>
<td>0.41/270</td>
<td>3.05</td>
<td>2.31</td>
<td>0.284</td>
</tr>
</tbody>
</table>

Table 14.2: Drag coefficients, pressure drops and Strouhal numbers for different variants of the laminar flow around a cylinder benchmark [174]. The Mach number is listed in the second column, the coarsest and finest grid resolution used in the simulation can be found in the third and fourth column.

The prediction of the pressure drops over the obstacle are in good agreement in the two-dimensional scenarios and found to be within a 7% interval. For the 3D1Q, slightly higher deviations from the reference data are observed for the pressure drop in Tab. 14.2. Further experiments have been conducted in this case, adapting the adaptive grid to exactly resolve the geometry of the cuboid-shaped obstacle. The respective simulation data showed a deviation of approx. 5% in the pressure drops.

The drag coefficients are found to lie within a tolerance interval of 5-6% around the reference data for the two-dimensional simulation setups. For the 3D-scenarios, the maximum deviation of 7% was only found on the coarsest grid level. The Strouhal number is also captured correctly, with $\leq 5\%$ deviation from the reference data.

The used scheme provides first-order accurate solutions with respect to the macroscopic flow data. With this respect, the obtained results from the benchmark suite are highly satisfying and provide a sufficient base for the future experiments. They also validate the drag force accumulation in particle scenarios since the same force accumulation procedure is used within the Peano code for both cylinder and particle simulations. An extension to second-order adaptive schemes as mentioned in Sec. 10.5 and the incorporation of boundary conditions of same order—yielding further improvements in the overall accuracy—are considered to be part of future work.

In the following, the performance of the Peano-based Lattice Boltzmann implementations is investigated. All measurements of the single-core performance were conducted on the Intel Core i7-architecture (i7-870, 2.93GHz, 8MB shared L3 cache, four 256kB L2 caches, four 32kB data L1 and 32 kB instruction caches). As mentioned in Sec. 10.2, the stack concept together with the space-filling curve approach of Peano yields high cache efficiency and respective computational efficiency in case of numerical schemes of high local compu-
tational loads and lower memory footprints. Especially the latter is not the case for the memory-intensive Lattice Boltzmann methods. In order to evaluate the influence of the copy operations between the stacks on the performance of Lattice Boltzmann solvers, three different Lattice Boltzmann implementations are considered:

- **Native:** this implementation\(^3\) is embedded in Peano in the native way that Peano applications are developed [125]. One Peano vertex contains one set of particle distribution functions. These distributions are stored directly within the vertex data structure and thus are copied between the different Peano stacks when traversing the grid. The evaluation of the Lattice Boltzmann scheme is performed in a cellwise manner, i.e. each Peano cell logically corresponds to one lattice node.

- **Block-vertex:** based on the Lattice Boltzmann core implementation that has been optimised for static spatially adaptive scenarios, the Lattice Boltzmann simulation is applied to block-structured grids as described in Sec. 10.3. The block size is chosen to be \( N = 6 \). Each vertex data structure contains the block of particle distributions so that the blocks are copied between the different Peano stacks during the grid traversal.

- **Block-service:** based on the same Lattice Boltzmann code as the Block-vertex variant, the block-structured Lattice Boltzmann grid is stored in the `GridManagementService`. Identifiers are stored within the vertex structures to allow for blockwise look-up operations. As a consequence, the blocks of distributions are not involved in any Peano-internal stack operations.

The performance for the three variants has been measured in three-dimensional cavity flow simulations which represent the standard test case to investigate the maximum performance of LB codes. In all simulations, the D3Q19 velocity discretisation was used. Different mesh sizes and grid types (spacetree vs. regular grid) have been investigated, as well as the usage of different compilers (gcc vs. icc). The results are shown in Tab. 14.3. A common measure for performance of Lattice Boltzmann codes is defined in terms of *Mega Lattice Updates Per Second* (MLUPS). It defines the number of lattice nodes that are treated by the collide-stream operation in one second. For adaptive simulations, however, this measure is not completely consistent: referring to the volumetric approach from Sec. 5.4 and the overlap region, it is only the coarse grid cells in which the collision step is executed, whereas both fine and coarse grid cells are involved in the streaming process. In the following, the determination of the MLUPS only considers the stream-collide process on the coarse grid cells of the overlap, as well as all non-overlap cells. Abstaining from the consideration of the fine grid cells in the overlap, this definition therefore builds a pessimistic measure for the performance on adaptive grids. All simulations were conducted for grid sizes which yield a runtime per (fine grid) time step which is smaller than five seconds. This guarantees the suitability of the underlying setup in terms of overall runtime\(^4\).

From Tab. 14.3, it is found that the **Native** implementation is by far the slowest, due to the massive amount of copy operations and additional overhead that is caused by the computation of the space-filling curve traversal order. Compared to the **Block-vertex** implementation, its performance is deteriorated by a factor of \( 4 - 5 \). Going from the **Block-vertex** to the **Block-service** implementation, the copy operations between the stacks are completely removed, except for a minimal amount of memory per Peano vertex which is required by the `GridManagementService` and the Peano kernel. The reduction in memcpy-operations results in a speed-up of approx. \( 4 - 5 \). This results in a maximum performance of the **Block-vertex** code on the spacetree grid of 5.20 MLUPS, considering the non-adaptive case. Another comparison was drawn between the spacetree-variant of the **Block-vertex** implementation and the analogous simulation using the regular grid implementation of Peano. This comparison allows to study the influence of the grid management by the Peano kernel:

\(^3\)The native Lattice Boltzmann implementation is developed in Peano, version 1. Slight differences with respect to performance may be noted when porting it to Peano, version 2. Previous experiments, however, showed that both versions show a similar level of performance with respect to the grid traversal.

\(^4\)A respective single-core simulation hence allows to compute \( O(10^5) \) time steps in less than a week which is assumed to be acceptable for large non-parallel scenarios.
### Table 14.3: Performance (in MLUPS) for three-dimensional cavity simulations using different Peano-based Lattice Boltzmann implementations.

<table>
<thead>
<tr>
<th>Implementation, grid type, compiler</th>
<th>Coarsest res., tree level</th>
<th>Finest res., tree level</th>
<th>No. cells/coarse grid time step</th>
<th>MLUPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native, spacetree, gcc</td>
<td>1/9, L2</td>
<td>1/9, L2</td>
<td>729</td>
<td>0.0461</td>
</tr>
<tr>
<td>Native, spacetree, icc</td>
<td>1/9, L2</td>
<td>1/9, L2</td>
<td>729</td>
<td>0.0480</td>
</tr>
<tr>
<td>Block-vertex, spacetree, gcc</td>
<td>1/18, L2</td>
<td>1/18, L2</td>
<td>5,832</td>
<td>0.188</td>
</tr>
<tr>
<td>Block-vertex, spacetree, icc</td>
<td>1/18, L2</td>
<td>1/18, L2</td>
<td>5,832</td>
<td>0.187</td>
</tr>
<tr>
<td>Block-service, spacetree, gcc</td>
<td>1/18, L2</td>
<td>1/18, L2</td>
<td>5,832</td>
<td>1.39</td>
</tr>
<tr>
<td>Block-service, spacetree, icc</td>
<td>1/18, L2</td>
<td>1/18, L2</td>
<td>5,832</td>
<td>1.40</td>
</tr>
<tr>
<td>Block-service, regular, gcc</td>
<td>1/18</td>
<td>1/18</td>
<td>5,832</td>
<td>2.50</td>
</tr>
<tr>
<td>Block-service, regular, icc</td>
<td>1/18</td>
<td>1/18</td>
<td>5,832</td>
<td>2.76</td>
</tr>
<tr>
<td>Native, spacetree, gcc</td>
<td>1/27, L3</td>
<td>1/27, L3</td>
<td>19,683</td>
<td>0.122</td>
</tr>
<tr>
<td>Native, spacetree, icc</td>
<td>1/27, L3</td>
<td>1/27, L3</td>
<td>19,683</td>
<td>0.126</td>
</tr>
<tr>
<td>Block-vertex, spacetree, gcc</td>
<td>1/54, L3</td>
<td>1/54, L3</td>
<td>157,464</td>
<td>0.608</td>
</tr>
<tr>
<td>Block-vertex, spacetree, icc</td>
<td>1/54, L3</td>
<td>1/54, L3</td>
<td>157,464</td>
<td>0.605</td>
</tr>
<tr>
<td>Block-service, spacetree, gcc</td>
<td>1/54, L3</td>
<td>1/54, L3</td>
<td>157,464</td>
<td>2.97</td>
</tr>
<tr>
<td>Block-service, spacetree, icc</td>
<td>1/54, L3</td>
<td>1/54, L3</td>
<td>157,464</td>
<td>3.08</td>
</tr>
<tr>
<td>Block-service, regular, gcc</td>
<td>1/54</td>
<td>1/54</td>
<td>157,464</td>
<td>3.41</td>
</tr>
<tr>
<td>Block-service, regular, icc</td>
<td>1/54</td>
<td>1/54</td>
<td>157,464</td>
<td>3.62</td>
</tr>
<tr>
<td>Native, spacetree, gcc</td>
<td>1/81, L4</td>
<td>1/81, L4</td>
<td>531,441</td>
<td>0.175</td>
</tr>
<tr>
<td>Native, spacetree, icc</td>
<td>1/81, L4</td>
<td>1/81, L4</td>
<td>531,441</td>
<td>0.186</td>
</tr>
<tr>
<td>Block-vertex, spacetree, gcc</td>
<td>1/162, L4</td>
<td>1/162, L4</td>
<td>4,251,528</td>
<td>1.09</td>
</tr>
<tr>
<td>Block-vertex, spacetree, icc</td>
<td>1/162, L4</td>
<td>1/162, L4</td>
<td>4,251,528</td>
<td>1.08</td>
</tr>
<tr>
<td>Block-service, spacetree, gcc</td>
<td>1/162, L4</td>
<td>1/162, L4</td>
<td>4,251,528</td>
<td>4.83</td>
</tr>
<tr>
<td>Block-service, spacetree, icc</td>
<td>1/162, L4</td>
<td>1/162, L4</td>
<td>4,251,528</td>
<td>5.20</td>
</tr>
<tr>
<td>Block-service, regular, gcc</td>
<td>1/162</td>
<td>1/162</td>
<td>4,251,528</td>
<td>4.91</td>
</tr>
<tr>
<td>Block-service, regular, icc</td>
<td>1/162</td>
<td>1/162</td>
<td>4,251,528</td>
<td>5.34</td>
</tr>
<tr>
<td>Block-vertex, spacetree, gcc</td>
<td>1/18, L2</td>
<td>1/54, L3</td>
<td>392,392</td>
<td>0.517</td>
</tr>
<tr>
<td>Block-vertex, spacetree, icc</td>
<td>1/18, L2</td>
<td>1/54, L3</td>
<td>392,392</td>
<td>0.508</td>
</tr>
<tr>
<td>Block-service, spacetree, gcc</td>
<td>1/18, L2</td>
<td>1/54, L3</td>
<td>392,392</td>
<td>2.62</td>
</tr>
<tr>
<td>Block-service, spacetree, icc</td>
<td>1/18, L2</td>
<td>1/54, L3</td>
<td>392,392</td>
<td>2.69</td>
</tr>
<tr>
<td>Block-vertex, spacetree, gcc</td>
<td>1/18, L2</td>
<td>1/162, L4</td>
<td>14,823,112</td>
<td>0.664</td>
</tr>
<tr>
<td>Block-vertex, spacetree, icc</td>
<td>1/18, L2</td>
<td>1/162, L4</td>
<td>14,823,112</td>
<td>0.658</td>
</tr>
<tr>
<td>Block-service, spacetree, gcc</td>
<td>1/18, L2</td>
<td>1/162, L4</td>
<td>14,823,112</td>
<td>3.20</td>
</tr>
<tr>
<td>Block-service, spacetree, icc</td>
<td>1/18, L2</td>
<td>1/162, L4</td>
<td>14,823,112</td>
<td>3.37</td>
</tr>
<tr>
<td>Block-service, spacetree, gcc</td>
<td>1/18, L2</td>
<td>1/486, L5</td>
<td>446,293,192</td>
<td>3.44</td>
</tr>
<tr>
<td>Block-service, spacetree, icc</td>
<td>1/18, L2</td>
<td>1/486, L5</td>
<td>446,293,192</td>
<td>3.66</td>
</tr>
</tbody>
</table>

since the spacetree grid implementation yields additional computational overhead to accomplish the stack-based spacetree grid traversal, the regular grid implementation is expected to yield faster computations. For the icc-compiled version on the finest non-adaptive grid (162³ cells), the performance increases from 5.20 to 5.34 MLUPS which represents approx. 2% of runtime.

Next, the spatially adaptive scenarios are considered. Comparing the performance of the adaptive setups to the performance of the non-adaptive simulations, the adaptive simulation is expected to be slower than the solver for the respective finest grid level in non-adaptive simulations: first, the additional streaming processes in the fine grid overlap regions that are not accounted for in the MLUPS definition from above yield a higher computational cost per lattice update. Second, compared to the non-adaptive simulation, less cells are traversed
yielding a lower performance—the respective increase in performance can be observed by considering the performance in the non-adaptive simulations for increasing grid resolutions. Third, the realisation of the adaptive algorithm and the storage scheme for the distributions have an additional impact on the performance: as the prolongation step requires the transport of the post-collision distributions from the coarser to the finer levels, the post-collision distributions cannot be streamed immediately, see point no. 2 from the subsection 10.5.1. The streaming step is therefore executed in the subsequent grid traversal. As a result, the respective distributions have to be touched a second time, resulting in additional load-store operations from/to memory. Hence, a performance drop due to the additional memory accesses is expected. The latter issue could be overcome by storing two sets of distributions and applying the standard A-B streaming pattern. In this case, the streaming could be accomplished immediately after the collision on the coarse levels. Besides, the prolongation could still be carried out as the post-collision distributions are still available from the first set of distributions. However, the memory requirements would hence be doubled which also may yield severe restrictions on the applicability of the method. This general problem—finding a compromise between memory and computational efficiency—is to be considered again within this chapter when addressing the overall memory requirements of the present Lattice Boltzmann application.

Considering the performance numbers from Tab. 14.3, it is found that the adaptive Lattice Boltzmann scheme (Block-service, spacetreegrid, icc) shows a performance of 87%/ 65% in the L2-L3/ L2-L4 case, compared to the non-adaptive L3/ L4 simulation. For the L2-L5 simulation, the performance is measured to be approximately 70%, compared to the finest non-adaptive (L4) simulation.

The performance of the Lattice Boltzmann application has further been compared to the open-source LB codes Palabos and OpenLB that are widely used in the LB community. For this purpose, the standard cavity scenario is considered using a $162 \times 162 \times 162$ grid and the D3Q19 discretisation. The results are listed in Tab. 14.4. The performance of the Peano-based Lattice Boltzmann application lies between the performance of the two community codes.

Stepping towards parallel simulations, the (regular grid) Lattice Boltzmann solver has been extended according to Peano’s callback structure to MPI-based distributed simulations. The domain decomposition applied in Peano is sketched in Fig. 14.2: the Peano cells are split among the processes whereas the vertices—denoted by coloured circles—are duplicated on the process boundaries. They are exchanged between the processes, and the respective vertex information can be merged using the mapping-specific `mergeWithNeighbour(currentVertex, neighbourVertex)` callback-method. This method is called locally for each copy `neighbourVertex` of the current vertex that exists on another process. Due to the duplication of the vertices, the solution in the respective LB blocks of these vertices has to be computed for each process which holds the respective copy.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Compiler</th>
<th>MLUPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block-service, spacetree</td>
<td>gcc</td>
<td>4.83</td>
</tr>
<tr>
<td>Block-service, spacetree</td>
<td>icc</td>
<td>5.20</td>
</tr>
<tr>
<td>Palabos</td>
<td>gcc</td>
<td>8.18</td>
</tr>
<tr>
<td>Palabos</td>
<td>icc</td>
<td>8.98</td>
</tr>
<tr>
<td>OpenLB</td>
<td>gcc</td>
<td>2.61</td>
</tr>
<tr>
<td>OpenLB</td>
<td>icc</td>
<td>4.48</td>
</tr>
</tbody>
</table>

Table 14.4: Performance in MLUPS for different Lattice Boltzmann codes in three-dimensional cavity scenarios using the D3Q19 velocity discretisation. The optimised Lattice Boltzmann implementation of Peano (Block-service, spacetree) is compared to the community codes Palabos and OpenLB.

---

5http://www.palabos.org
6http://www.openlb.org
Different strategies for merging the information of duplicated blocks exist; two strategies have been evaluated within the scope of this thesis. The first approach consists in the method of difference tracking; therefore, consider again Fig. 14.2: if particle distribution functions would need to leave “proc 1” as sketched in this figure, the non-local, that is Peano cellwise, streaming operation from the subsection 10.4.3 cannot be performed for these distribution functions on this process. The distributions will thus be bounced back by the cell-local swapping procedure and remain—except for local modifications such as the collide step—unchanged. Tracking the difference between the post-collision distributions and their respective bounced-back counterparts consequently allows to identify the distribution functions that could not be streamed due to a process boundary. If the respective difference in the post-collision states is transferred between the processes, the difference can just be added on top of the bounced-back distribution functions. This method has been tested for all velocity discretisation schemes, and the weak scaling in a cavity scenario is shown in Fig. 14.3 on the left. The Peano vertices do not have any knowledge on their spatial location. As a consequence, the differences for all particle distribution functions at the local block boundary need to be sent to all neighbouring processes although only subsets of the respective boundaries might be required by each respective neighbour. This induces severe communication overheads on core counts $> 1024$ (2D) and $> 8$ (3D) assuming an efficiency of 90% to be still acceptable. An improved communication scheme is hence required. A possible solution is sketched in the following.

In the Figure 14.2, the LB cells which are contained in the local Peano cells of each process and which belong to the duplicated vertices on the process boundary are coloured in red.
Table 14.5: Memory requirements in Peano-based LB simulations. The second column shows the bytes that are required per Lattice Boltzmann cell, the third column shows additional bytes for each Lattice Boltzmann block/ Peano vertex. The following columns show the total memory requirements for different block sizes.

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Bytes/ LB cell</th>
<th>Bytes/ Peano vertex</th>
<th>kB/ LB block size B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$B = 3$</td>
</tr>
<tr>
<td>D2Q9</td>
<td>98</td>
<td>5</td>
<td>0.866</td>
</tr>
<tr>
<td>D3Q15</td>
<td>154</td>
<td>9</td>
<td>4.07</td>
</tr>
<tr>
<td>D3Q19</td>
<td>186</td>
<td>9</td>
<td>4.91</td>
</tr>
<tr>
<td>D3Q27</td>
<td>250</td>
<td>9</td>
<td>6.60</td>
</tr>
</tbody>
</table>

(process 1) and green (process 2). Assume the non-coloured cells of the vertices on the boundary of process 1 to contain valid information at time step $n$ of the LB algorithm and assume the MPI communication to be switched off. Since the distributions may travel at most one cell per time step in the default LB algorithm, all LB cells except for the ones of the outermost cell layer (that is the LB cells at the very right of the domain of process 1) contain valid information in time step $n + 1$. Another two time steps can be performed locally, still yielding a valid distribution field in the red-coloured domain. The same argument holds for the green-coloured domain on process 2. Hence, after three time steps, we still have a globally valid distribution field except for the non-coloured “overlap layer” on the process boundary. As a consequence, it is sufficient to communicate the missing information between the processes every three time steps only. Concluding, the communication is switched off for $N/2$ LB time steps where $N$ represents the block size of one LB block per Peano vertex. After $N/2$ time steps,

1. the red-coloured LB cells are sent from process 1 to process 2, and the green-coloured cells are sent from process 2 to process 1
2. the green-coloured cells are received on process 1 from process 2, and the red-coloured cells are received on process 2 from process 1
3. the overlap layer, that is the non-coloured cells of the vertices on the process boundary, is overwritten by the received LB cells.

This parallelisation approach—based on the idea of domain overlapping—has been implemented and tested for the D2Q9 model in analogous cavity scenarios. The weak scaling is shown in comparison to the difference tracking approach in Fig. 14.3 on the right. An increase in efficiency of 10% is observed for 4096 processes in case of the overlap approach. As a last point, the memory requirements of the Peano-based Lattice Boltzmann implementation shall be addressed. The memory required per Lattice Boltzmann cell and per Peano vertex is listed in Tab. 14.5. Depending on the level of granularity in the local grid refinement, bigger or smaller block sizes may be preferable. For this reason, different block sizes ($B \in \{3, 6, 12, 18, 24\}$) are considered in Tab. 14.5. Besides the density (1 double), velocity ($D$ doubles) and the distributions ($Q$ doubles) that are stored in each LB cell\(^7\), two additional bits are stored for the cell to describe whether the cell is located in- or outside the computational domain and whether it is close to a computational boundary. This yields a total amount of $2 + 8(D + 1 + Q)$ bytes per cell\(^8\). In order to control the streaming process in dynamic fluid-structure scenarios, each Peano vertex holds $1 + 2^D$ bytes—one byte to decide whether density interpolations may be required inside the respective LB block and $2^D$ bytes to determine whether the non-local streaming step procedure has already been carried out over the respective Peano cell.

---

\(^7\)The Lattice Boltzmann implementation that is optimised for static grids even requires only the storage of the $Q$ distributions. Macroscopic quantities are omitted in this case.

\(^8\)No particular compressed data formats are used, such as DaSGen structures from the Peano records.
Due to these low memory requirements, the four-level adaptive simulation reported in Tab. 14.3, going from level 2 to level 5, can be carried out, using 4.48 GB of memory\(^9\) at acceptable runtime. With the machine having 8GB, doubling the distribution set in order to gain even higher performance—due to a merged collide-stream implementation for the coarse grid levels—would yield memory requirements of 8.14 GB. Besides, the same scenario could hardly be solved by a non-adaptive Lattice Boltzmann solver on an ordinary desktop machine. With a resolution of \(486^3\) cells, even the storage of a single set of distribution functions would result in memory requirements of more than 16 GB! In order to obtain the solution in the same amount of time on a single core, a performance of 25 MLUPS would be required. This could only be obtained on nowadays hardware using vectorisation and hardware-aware optimisation techniques, cf. [74].

Concluding this chapter, the Lattice Boltzmann application has been validated by typical cavity and channel benchmarks. The respective results show very good agreement with the reference solutions. For the following discussions, the adaptive Lattice Boltzmann solver therefore provides a sufficient level of accuracy. Still, a second-order accurate adaptive scheme is desirable to further increase accuracy. This step is considered to be part of future work. In terms of performance, it has been demonstrated that the development of a service-based grid management which provides whole blocks of data for each Peano vertex is necessary to avoid Peano’s copy operations between the stacks and to reduce the traversal costs of the spacetree. The same block-concept is applicable to other applications with similar memory requirements as the LB method or to applications for which a block-structured refinement is sufficient. One example is given by PeanoClaw\(^0\) which combines the framework ClawPack for hyperbolic solvers and Peano: it hence allows for Peano-based adaptive mesh refinement and local time stepping for the ClawPack-based solvers. The present Lattice Boltzmann implementation shows a similar performance in terms of MLUPS as well-known community codes. The performance of the LB code could be further increased by using a second set of distributions. However, several scenarios—such as the four-level adaptive cavity simulation—may not be computable on a single core in this case. The same holds for more complex Lattice Boltzmann implementations such as Lattice Boltzmann-based multicomponent codes which require the storage of at least one distribution set per fluid component. A respective simulation code is arising from the presented Lattice Boltzmann application within Peano and is under current development. Besides, vectorisation and related optimisation techniques are expected to yield further improvements in performance. The major goal of this thesis, however, consists in the exploration of fluid dynamic processes on different scales and the incorporation of respective functionalities rather than pure Lattice Boltzmann code optimisation. With the performance of the current LB code comparable to other well-established LB codes, its performance is found to be sufficient for the following computations at this stage. Further tunings of the sequential algorithm are subject of future developments. First steps have been taken towards the distributed parallelisation of the Lattice Boltzmann application. The approach based on the overlap structure of the process boundaries appears to be most promising. Still, improvements are required to obtain a scalable code on bigger core counts. For example, the Peano vertex records which solely hold the block identifier for the corresponding LB block are currently automatically communicated between the processes by the Peano kernel in every time step although a communication every \(N/2\) time steps (\(N\) is the LB block size) would be sufficient. Peano currently does not provide the feature to switch communication of the records completely off yet. This induces additional communication overhead. Future works into this direction and towards the parallelisation of the spatially adaptive Lattice Boltzmann scheme are planned.

\(^9\)For the optimised implementation, 3.78 GB are required.
\(^0\)http://www5.in.tum.de/peano/src/sec/peano/applications/peanoclaw/
peanoclaw0_dir_description.html
15 Flows at Finite Knudsen Numbers: Validation and Microreactor Simulations

In the following, the results for finite Knudsen number flows which are based on the theoretical extensions from Sec. 5.7 are presented. Most of the results have been published in [138].

First, the Lattice Boltzmann application is validated for slip flows. Two-dimensional pressure-driven channel flows are set up according to the descriptions from Verhaeghe et al. [177], using a $1100 \times 10$ non-adaptive grid. In order to have a “perfect” comparison for validation, the same pressure extrapolation boundaries have been applied in the simulations as reported in the respective works by Verhaeghe et al. [177]. The results for stream- and spanwise velocity components along the cross-section as well as the pressure deviation from its linear distribution along the centreline of the channel are shown in Fig. 15.1 for Knudsen numbers $Kn = 0.0194$ and 0.388. All profiles obtained from the Peano-based Lattice Boltzmann simulations and the ones reported in [177] perfectly match, indicating the correctness of the Peano implementation. For $Kn = 0.388$, a significant deviation in the pressure profile of both the Navier-Stokes and the Lattice Boltzmann solvers from the DSMC-results can be observed. The streamwise velocity profiles agree well for all solvers in this Knudsen number range.

Next, the Lattice Boltzmann solver is validated for the transition regime by comparing the velocity profile to the results obtained by Li et al. [114]. The underlying setup resembles a two-dimensional force-driven periodic channel flow. For this test, the periodic boundary conditions that are required at the left and right channel boundaries have been integrated into the Peano-based Lattice Boltzmann application. The forcing term which is required to drive the periodic flow in x-direction is incorporated as an additional term into the multiple-relaxation-time operator following the procedure in [114]. The underlying concept resembles the incorporation of additive Brownian forces, cf. Sec. 5.3. The boundary conditions at the channel walls are implemented according to Eqs. (5.36), (5.38). Besides, the viscosity adjustment discussed in Sec. 5.7 is applied in the simulations. Figure 15.2 shows the arising profiles for different Knudsen numbers $Kn \in [0.1128; 4.5135]$. Going from moderate to higher Knudsen numbers, the graphs clearly show the strong deviation of the second-order slip Navier-Stokes solution from the results obtained in [143] by solving the linearised Boltzmann equation. In contrast, the Lattice Boltzmann implementations still capture the correct profile, due to their kinetic origin and close relation to the linearised Boltzmann equation. Besides, the Peano-based implementation again perfectly matches the reference solution of [114].

Li et al. also pointed out the capability of their model for transition flows to capture the Knudsen minimum phenomenon: Knudsen [99] discovered that for a given channel flow, the mass flux first decreases when increasing the Knudsen number. After reaching a minimum at $Kn \approx 1$ [98], the mass flux slightly increases again. The respective simulation scenarios from [114] to investigate this phenomenon by means of the Lattice Boltzmann extension was confirmed, using the Peano-based implementation. The result is shown in Fig. 15.3: the Knudsen minimum is predicted at $Kn \approx 0.9$ by the Lattice Boltzmann method for transition flows, using second-order velocity slip conditions near the walls and viscosity adjustment.

Going to three-dimensional scenarios, the Peano-based Lattice Boltzmann solver is validated by considering the pressure-driven duct flow scenario and comparing the results to those obtained by Tang et al. [169] and Colin and Aubert [29]. Due to the sensitivity of the pressure deviation in two-dimensional scenarios, the pressure deviation along the centreline of the duct is also measured in this test; for velocity profiles, see the descriptions in the following application of the finite Knudsen LB solver to microreactor geometries. The results for the pressure profile are compared with the results of Tang et al. [169] and Colin and Aubert [29]. Tang et al. [169] used a BGK-based Lattice Boltzmann implementation, together with respective slip boundary conditions to compute the duct flow. Colin and Aubert [29] proposed an analytical model for respective gas flows in microducts. Simulations were carried out for height-to-width ratios $H/W = 1$ and $H/W = 0.25$ and a resulting regular Cartesian grid discretisation of $1200 \times 24 \times 24$ or $1200 \times 48 \times 48$, respectively. Pressure in- and
Figure 15.1: Velocity and pressure values in pressure-driven channel flows at Knudsen numbers \( \text{Kn} = 0.0194 \) and \( 0.388 \) in the slip flow regime. Top row: pressure deviation from the linear distribution along the centreline. Mid row: normalised streamwise velocity profile along the channel cross-section. Bottom row: normalised spanwise velocity profile along the channel cross-section. The Peano-based results are shown by solid red lines, the results from [177] are given by black triangles, the first-order slip Navier-Stokes solutions [177] are depicted by green diamonds and IP-DSMC simulation results [160] are visualised by blue circles.

Outlet ratios \( p_{in}/p_{out} = 1.94, 2.37 \) and \( 2.64 \) were imposed, and the Knudsen number was set to \( \text{Kn} = 0.055 \). The first-order slip boundary conditions were applied at the channel walls, together with the viscosity adjustment rule in the whole duct. Further extensions in terms of boundary conditions are required for this kind of scenario: due to edge-like boundaries, the boundary normal is not prescribed uniquely anymore for the neighbouring Lattice Boltzmann cells. In this case, the standard bounce-back scheme is applied to the respective distributions, so that their mass flux vanishes. A slight reduction in the boundary slip velocity near the edges is consequently expected. The pressure deviation from the linear profile along the centreline of the duct for the different simulations is provided in Fig. 15.4. In all scenarios, the Peano-based solution is very close to the results of the other two re-
Figure 15.2: Normalised streamwise velocity profiles in force-driven periodic channel flows for different solvers at various Knudsen numbers in the transition flow regime. The present Peano-based Lattice Boltzmann implementation with second-order boundary conditions and viscosity adjustment is shown by the solid red line, the Lattice Boltzmann implementation proposed by Li et al. [114] is depicted by black triangles, the second-order slip velocity Navier-Stokes solution by Hadjiconstantinou [75] is visualised by green diamonds and the solution of the linearised Boltzmann equation by Ohwada et al. [143] is given by blue circles.

For the height-to-width ratio \( H/W = 0.25 \), the present LBM shows a slightly smaller deviation in the pressure. The respective curve for the pressure deviation is still very close to the BGK-based simulation results from Tang et al., indicating the correctness of the approach used in the Peano-based Lattice Boltzmann application.

Having validated the finite Knudsen-extension of the Peano-based Lattice Boltzmann application, the simulation software is ready to be used for realistic flow scenarios. However, only few publications dealt with the simulation of respective scenarios [156]: Colosqui et al. [30] applied a BGK-based finite Knudsen-code to the simulation of electro-mechanical resonators, and Tang et al. performed various flow studies for two- and three-dimensional porous media [167, 169]. To the author’s knowledge, no adaptive Lattice Boltzmann simulations for finite Knudsen problems have been reported yet. In the following, the Peano-based
Figure 15.3: Dimensionless mass flux in a periodic force-driven channel flow at different Knudsen numbers. Left: wide range of Knudsen numbers. Right: close-up on Knudsen number regime at $Kn \approx 1$. The Peano-based Lattice Boltzmann implementation with second-order slip boundary conditions and viscosity adjustment is shown by solid red lines, the results by Li et al. [114] are given by black triangles. A respective Navier-Stokes plot by Hadjiconstantinou is given in form of green diamonds, and a solution of the Boltzmann equation by Cercignani et al. is shown by blue circles. All data—except for the Peano-based solution—have been extracted from [114] for the sake of comparison and validation.

Figure 15.4: Pressure deviation along the centreline in different pressure-driven duct flow scenarios. Left: $H/W = 1$. Right: $H/W = 0.25$. The Peano-based Lattice Boltzmann solution is shown by solid lines, the LB implementation proposed by Tang et al. [169] is given by black triangles and the analytical solution by Colin and Aubert [29] is given by black circles.

Lattice Boltzmann implementation is used to solve the flow within a complex duct system of a microreactor, applying static spatial adaptivity and coarse-graining of the collision process in terms of a mixed MRT-BGK problem formulation. The underlying geometrical setup is schematically drawn in Fig. 15.5, resembling the descriptions in [113]: on the left, the fluid enters the duct system and streams through the ducts into the reactor chamber where for example oxidation processes may take place to remove pollutants from the fluid. Afterwards, the fluid leaves the chamber via a second duct system and leaves the simulation domain on the right side. Analogously to the problem of having edge-like boundaries for the 3D-channel scenario, it is also corner-like boundaries that can occur in the microreactor setup. Here, also the half-way bounce back scheme is used for the distributions. As the

\[11\] Although the microreactor discussed in [113] is used for water treatment, similar geometries are also imaginable for rarefied gas flows.
number of corners in the overall simulation domain is very small compared to the planar boundaries, the application of no-slip conditions instead of slip conditions is expected to have a minor influence on the overall simulation outcome.

Different two-dimensional experiments were conducted in this duct geometry, applying a two-level static adaptive grid together with the volumetric Lattice Boltzmann formulation from Sec. 5.4: the coarse grid is used in the inner part of the reaction chamber whereas the fine grid is applied near all geometrical boundaries and the ducts, cf. Fig. 15.5 on the top. The second-order slip-boundary condition is applied at all walls of the chamber and the ducts, together with the viscosity adjustment rule and the respective MRT collision operator. This yields Knudsen numbers in the range of 0.11 to 0.21 in the ducts and the chamber. As the application of the half-way bounce back rule at corners may yield a reduction in wall slip, the mass flux before and after branching in the duct system was investigated. For example, if $m_i$ denotes the mass flux at checkpoint $x_i$ (see Fig. 15.5 on the lower left for the different checkpoint marks), then it needs to hold that $m_1 = m_2 + m_3$ and $m_2 = m_4 = m_6 + m_7$. It was found that the deviations in the mass flux are below 0.4% in each branch. Besides, the profiles in the ducts were examined and compared to the solution of pure channel-flow problems on non-adaptive grids as discussed previously in this chapter. The profile comparisons are given for various checkpoints in Fig. 15.6, including results from a microreactor simulation using non-adaptive grids. All simulation results agree very well. The maximum pointwise deviation in the streamwise profile from the plane-channel solution for the respective checkpoints has been measured and found to be 1.2% in both adaptive and non-adaptive case. The speedup—going from the non-adaptive to the adaptive simulation of the duct system—was measured to be 1.154. Only a small speedup is expected in this case since the reactor chamber is modelled to be quite narrow, cf. Fig. 15.5. As a consequence, only a small sub-region of the overall geometry is resolved by coarser grid cells.
Figure 15.6: Comparison of streamwise velocity profiles at different checkpoints of the two-dimensional duct system. The reference channel profile is given by the black line. The adaptive Lattice Boltzmann solution in the duct system using the MR T operator on all grid levels is given by red marks, the hybrid MR T-BGK approach is depicted by blue symbols. The non-adaptive simulation results for the microreactor geometry is drawn by green squares.

In order to further reduce computational costs, a hybrid MRT-BGK collision approach has been investigated. The main motivation to apply the MRT collision model was to suppress artificial slip effects near walls. However, in the inner part of the reaction chamber, the impact of the walls is expected to play a minor role. Hence, the application of the computationally cheaper BGK collision model on the coarse grid within the reactor chamber should yield a smaller runtime and still preserve the characteristics of the flow. The respective scenario was simulated, using MRT in all fine grid cells and BGK in all coarse grid cells; the results are also included in Fig. 15.6. The streamwise velocity profiles agree well with the ones from the other simulations, with a maximum pointwise deviation from the plane-channel solution of 1.7%. In terms of performance, the hybrid MRT-BGK approach yielded a further reduction in runtime of 1.5%.
Finally, a three-dimensional microreactor was simulated. A non-adaptive grid consisting of $240^3$ cells was used, together with the first-order slip-boundary conditions and viscosity adjustment. The size of the reactor was chosen to yield Knudsen numbers of 0.017 and 0.0052 in the duct system and the reactor chamber, respectively. Similar to the two-dimensional case, the mass flux error before and after duct branching as well as the streamwise velocity profiles at the different checkpoints were examined. The mass flux error did not exceed 0.4%. The velocity profiles are depicted in Fig. 15.7. They are again found to be in good agreement in both pure channel and microreactor simulation.

Concluding, the Peano-based Lattice Boltzmann application has successfully been extended to the slip and transition flow regime, allowing for the simulation of rarefied gas flows over a wide range of Knudsen numbers. Besides the capability of solving two- and three-dimensional problems, the spatially adaptive simulation of respective flow scenarios is immediately possible. The latter has been used to deal with a complex duct system that is connected to a microreactor chamber: due to the spatial adaptivity, a coarse-graining of the computational domain in the chamber was established, yielding reduced computational costs. Higher speedups than in the current example are expected for large-sized chambers where spatial adaptivity significantly pays off. Besides the spatial adaptivity, a coarse-graining of the collision process was applied switching to the cheap BGK collision kernel when slip effects—and thus the application of the MR T collision operator—play a negligible role. A small speedup was measured using this procedure while still obtaining acceptable simulation results. Again, higher gains in runtime are expected when larger domains can be coarse-grained. The simplification of the collision step may also strongly reduce the runtime when the fine- and coarse-grained collision model show a more significant difference in computational costs. A respective example is discussed in Sec. 16.1.
Typical application areas for Lattice Boltzmann methods comprise simulations in complex flow geometries or flows with suspended particles. In the following, the particle translocation inside a drift ratchet is discussed. The respective results have been previously published in [137].

Drift ratchets are micro- to nano-sized channel-like structures that are characterised by a varying diameter, changing along its longitudinal axis, cf. Fig. 16.1. It has been observed that particles that are released in an oscillatory flow field inside such a ratchet may start to drift into a particular direction, although, on the macroscale, one would expect that the particle remains within a fixed interval, periodically moving forward and backward. Amongst others, Brownian motion effects have been found to play a crucial role in these kinds of scenarios, see for example [77] for an extensive review on different types of ratchets and Brownian motors. The simulation of the underlying phenomenon, however, is very complex: first, a code for fluid-structure phenomena is necessary to capture the interaction of the particle and the fluid. Second, the respective particle drifts may only be observed over very long time intervals. With the ratchets typically operating in the diffusive limit, that is at $Re \ll 1$, the time step $dt$ in fluid dynamics simulations scales with the squared mesh size $dx$, $dt \propto dx^2$. Hence, a huge number of time steps is required, yielding the necessity of efficient, parallel and (spatially) adaptive simulation codes.

In the following, two simulation approaches to address the described flow scenario are discussed: in Sec. 16.1, the developed Lattice Boltzmann application is used to predict the movement of an isolated particle. Here, the coupling of the flow solver and the particle on dynamically changing grids is validated. Afterwards, the impact of thermal fluctuations on the diffusion of the particle is addressed. Within this context, a spatial cut-off mechanism for the fluctuations is discussed, allowing to significantly speed up the particulate flow simulation. Finally, both extensions—dynamic adaptivity and cut-off mechanism—are used in the simulation of the particle translocation inside a drift ratchet. Although these extensions allow for the efficient inclusion of Brownian effects into the flow simulation, the arising system still yields long simulation times in case of vanishing Reynolds numbers. In order to address longer time scales while still incorporating the important physical effects, a hybrid simulation method has been developed and is presented in Sec. 16.2. Here, an expert system is developed that allows for switching back and forth between two particle-fluid simulation approaches: one approach is given by the aforementioned LB-particle interaction code. The second method is provided by a Navier-Stokes solver which uses the Faxén correction in a
post-processing step to predict the behaviour of the suspended particle. With the latter method being applied on spatially coarser grids and neglecting explicit two-way couplings of the particle and the fluid, it uses significantly larger time steps than the LB-particle simulation. The hybrid approach thus allows for long-time particle simulations.

16.1 Short-Time Particle Displacement

Stepping towards the short-time simulation of the particle-in-ratchet scenario, the validation of the required ingredients shall be discussed first. The mapping of the particle geometry onto the grid and respective reflagging procedures including density interpolations near the particle (cf. Sec. 5.6) are validated in different unit tests, checking for every possible moving boundary location in two and three dimensions. The evaluation of forces acting on an obstacle has already been validated in the context of the different benchmark scenarios from Sec. 14. Besides these extensions of the LB scheme, Peano’s built-in structural solver for the isolated spherical particle has also been validated by respective unit tests.

Next, the validity of the dynamic adaptivity algorithm from the subsection 10.5.2 is investigated. Therefore, a spherical particle is simulated in a plain channel flow, cf. Fig. 16.2: once released, the particle is accelerated by the surrounding fluid and adapts to the flow velocity. As the lower Reynolds number regime is of major interest in the following, the Reynolds number is set to \( \text{Re} = 1 \). The scenario is simulated by the BGK-based LB method on a fully resolved fine grid and a dynamically changing grid. The latter grid is adapted over time such that the particle is always surrounded by the finest grid level. Both two- and three-dimensional cases are considered. In the 2D case, an adaptive grid which consists of three grid levels is used whereas a two-level grid is applied in the 3D case. The particle position, velocity and the force exerted by the fluid onto the particle are measured over time and are shown in Fig. 16.3. With the particle released in the very middle between both channel walls, the \( x \)-components of the position, velocity and force vectors remain unchanged throughout the simulation, and the considerations can therefore be restricted to the \( x \)-components. The dynamically adaptive solution agrees very well with the non-adaptive results for all considered quantities.

Next, the influence of spatial adaptivity on thermal fluctuations and, hence, the Brownian diffusion of the spherical particle is studied. From theory, it is well-known that the mean-squared displacement \( \langle x_p(t) \rangle \) of the particle scales quadratically in the short-time limit and linearly in the long-time limit, that is \( \langle x_p(t \to 0) \rangle - x_p(t = 0) \propto t^2 \) and \( \langle x_p(t \to \infty) \rangle - x_p(t = 0) \propto t \). Incorporating the fluctuations via the respective FLB collision model (cf. Sec. 5.3) in a non-adaptive LB simulation yields thermal noise in all LB cells which is uncorrelated in space and time. However, going to the adaptive case, correlations are automatically introduced, since the post-collision distributions within the coarse grid overlap cells are homogeneously distributed in the respective fine grid cells. So, the distributions that get streamed into the fine grid region show a high level of redundancy, if they originate from the same coarse grid cell. Besides, the fluctuations are automatically reduced on the coarser grid levels due to the coarser mesh sizes and time steps, cf. Eq. (5.18). Reviewing the thermal scaling factor for the distributions

\[
\Theta_{ii} \propto \sqrt{\frac{k_B T dt^2}{c_s^2 dx^2 D + 2}}, \tag{5.18}
\]

this scaling factor can be observed to decrease equally fast in 2D as the scaling factor for the viscous terms and slightly faster in 3D; remember that the scaling factor for the viscous terms is proportional to \( dt/dx^2 \), cf. Sec. 5.19. Following this discussion, the influence of spatial adaptivity on fluctuating simulations is not clear yet. Besides the decay of the fluctuations on coarser grid levels, it is also the additional computational costs induced by the fluctuating model that deserve closer consideration. The fluctuations in the collision model are modelled via Gaussian pseudorandom numbers. Depending on the underlying velocity space discretisation and the dimensionality of the problem

\( ^{12} \text{The characteristic length is chosen to be the particle diameter.} \)
(2D/3D), several pseudorandom numbers are required in each LB cell. For the D2Q9 model, one needs at least three random numbers per LB cell, that is one random number for each viscous stress mode in the multiple-relaxation-time formalism of the collision model. If particular tunings of the higher-order modes shall be used, another three random numbers are needed. In three-dimensional scenarios, this becomes even worse with 6-15 Gaussian pseudorandom numbers required per LB cell. Hence, the generation of the Gaussian pseudorandom numbers can become a computationally expensive part in the fluctuating LB algorithm. For an impression on this augmented computational complexity, Tab. 16.1 summarises the relative runtimes for different collide-stream kernels, comparing BGK-, MRT- and FLB-based kernels. The performance of the MRT-kernel which is based on the FLB-implementation is slower than what is expected for tuned MRT versions (see Sec. 5.3). The FLB implementation requires at least two matrix-vector multiplications (mapping the distributions to moment space and back or updating the standard MRT part and the random noise separately), whereas a tuned particular MRT implementation would require only one. Therefore, a tuned MRT implementation is expected to gain another $\approx 5 - 10\%$ in the example from Tab. 16.1. It can clearly be observed that the runtimes of the fluctuating approach are dominated by the random number generation, even in the case of minimal random number requirements which is shown in the column $t_{\text{min}}^{\text{FLB}}$. In the exemplary computations described in this table, the Polar method [121] has been used to generate the required pseudorandom
Figure 16.3: X-component of the position, velocity and force vector of an isolated spherical particle released in a pressure-driven channel flow. Once released, the particle is accelerated over time and adopts to the flow velocity. The yellow squares represent the solution from the non-adaptive simulations on a fully resolved fine grid whereas the blue lines show the results from the dynamically adaptive simulations. Left column: 2D-scenario. Right: 3D-scenario, based on the D3Q19 discretisation. Analogous results have been obtained for the D3Q15 and the D3Q27 scheme.

numbers. Similar results have also been obtained for other random number generation methods such as the Box-Muller method [16] or the Mersenne-Twister [122]. Recently, another strategy for the Gaussian random number generation has been developed by Ladd [105] which has been found to be fast and accurate in fluctuating LB simulations. With the random number generation representing a computationally expensive part and with the thermal fluctuations rapidly decaying on coarser grid levels, the idea emerged to
Table 16.1: Relative runtimes of different collide-stream implementations. The runtimes are normalised by the runtime of the BGK-based implementation. \( t^{BGK} \): BGK-based implementation. \( t^{MRT} \): MRT-scheme (based on the FLB-implementation). \( t_{min}^{FLB} \): FLB-scheme, generating the lowest number of Gaussian random numbers. \( t_{max}^{FLB} \): FLB-scheme, generating the maximum number of Gaussian random numbers.

<table>
<thead>
<tr>
<th>Velocity set</th>
<th>( t^{BGK} )</th>
<th>( t^{MRT} )</th>
<th>( t_{min}^{FLB} )</th>
<th>( t_{max}^{FLB} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2Q9</td>
<td>1.00</td>
<td>1.13</td>
<td>2.30</td>
<td>3.30</td>
</tr>
<tr>
<td>D3Q19</td>
<td>1.00</td>
<td>1.24</td>
<td>2.21</td>
<td>3.51</td>
</tr>
</tbody>
</table>

Figure 16.4: Mean-squared particle displacement for 2D particle simulations in an empty box over (long) (left) and shorter (right) time intervals. The latter is visualised in loglog-scaling, illustrating the quadratic dependency of diffusion over time.

introduce a cut-off mechanism for the fluctuations: if the grid resolution becomes coarser than a certain threshold, the fluctuations are so small that they may not yield significant contributions to the overall flow simulations anymore. In this case, one could either switch them off and use the standard MRT collision kernel or—as described and tested in the following—switch to the BGK collision kernel which comes at even smaller computational cost.

Therefore, this cut-off approach is investigated, considering a diffusing spherical particle in a box filled with fluid at rest. Different variants are compared:

- “Non-adaptive FLB”: the fluctuating collision model is applied on a non-adaptive, fully resolved grid.
- “Adaptive FLB”: a two-level grid is used where the fluctuating LB model is applied on each grid level.
- “Adaptive BGK-FLB”: a two-level grid is used where the BGK operator is applied on the coarse grid and the fluctuating collision operator is applied on the fine grid level.

The adaptive grid is refined near the boundaries of the box and near the particle, respectively. Its resolution corresponds to the grid resolution of the non-adaptive grid. The (dimensionless) level of the thermal fluctuations is chosen as \( k_B T = 9 \cdot 10^{-7} \) on the finest grid level. The three setups are solved in two dimensions; for each case, 50 samples are computed. The mean-squared displacement \( \langle x_\mu(t \to \infty) - x_\mu(t = 0) \rangle \) is evaluated from the samples and plotted in Fig. 16.4. The short-time diffusion is underestimated by the “Adaptive FLB” and “Adaptive BGK-FLB” scheme, due to the smoothing of the fluctuations on the coarser grid levels. The linear relation of the mean-squared displacement over longer time intervals is captured correctly.

Having compared and validated the methods for multi-level fluctuating hydrodynamics and dynamic adaptivity, the LB application is applied to the transport problem of a sub-micron particle inside a drift ratchet, see Fig. 16.5. As mentioned before, it has been shown that...
such particles may undergo a directed motion within these devices when exposed to oscillating pressure fields. Previous studies have been performed, investigating the influence of the pressure frequency and the initial particle positions [17, 18]; long time scales have been considered using fluid-structure approaches based on a flexible Navier-Stokes implementation. In the following, the methodology developed within this section is used to point out the importance of Brownian fluctuations in this scenario on the nanoscale. Therefore, the particle translocation inside the ratchet is considered in both fluctuating and non-fluctuating simulations. For this purpose the setup from Fig. 16.5 is used: the two chambers have a minimum and maximum (dimensionless) radius of 0.16 and 0.3, respectively. Each chamber has a length of 1.0. The spherical particle is chosen to have a density $\rho_s = 8.0 \cdot \rho_{\text{fluid}}$ where $\rho_{\text{fluid}}$ denotes the density of the fluid. The radius of the particle is set to 0.1. Three different initial positions for the particle are defined: $P_0 = (0.5, 0.5)$ is located right in the centre of the first pore chamber, $P_1 = (0.99, 0.5)$ lies between the two chambers and $P_2 = (0.5, 0.55)$ is again located in the centre of the first pore chamber, but slightly above the longitudinal symmetry axis of the 2D pore. For each initial position, three simulations are performed, using a two-level dynamically adaptive grid. The “BGK-BGK” setup represents a non-fluctuating simulation, applying the BGK collision operator on both grid levels. A completely thermalised scenario “FLB-FLB” is solved where the fluctuating LB model is used on the coarse and the fine grid. Besides, a “BGK-FLB” hybrid approach—applying the BGK model on the coarse grid and the FLB model on the fine grid—is simulated. The fluctuations are chosen to have a magnitude of $k_B T = 10^{-6}$ on the coarser grid level; they are scaled according to Eq. (5.18) on the fine grid. At the boundaries, velocity inlet and pressure outlet conditions are supplied; the flow is oscillating within the pore at a frequency of $f = \frac{1}{2 \pi \omega_{\text{osc}}}$ for the pressure oscillations. The maximum magnitude in flow velocity is chosen such that the Reynolds number emerges at $Re = 0.1$, based on the maximum integral velocity at the pore inlet. The simulations are conducted over $9 \cdot 10^5$ time steps. Following the theoretical considerations from the subsection 10.5.2, a dynamic grid update is sufficient to be carried out every 10 coarse grid time steps. This results in a computational overhead of approx. 2.5%.

The displacement of the spherical particle is determined and plotted over time in Fig. 16.6. In the non-fluctuating BGK-BGK cases $\vec{P}_0$, $\vec{P}_2$, the particle slightly drifts to the right. However, in the BGK-FLB and FLB-FLB simulations, the particle moves to the left. A similar behaviour can be observed in the fluctuating simulations for the initial position $\vec{P}_1$. In the BGK-BGK simulation, the particle remains within its periodic bounds in this case. The
Figure 16.6: Displacement of a spherical particle along the rotation axis of the drift ratchet for different initial particle positions. A pressure frequency of $f = \frac{1}{25000}$ is used in the simulations of the three upper graphs whereas $f = \frac{1}{10000}$ in the simulations of the last graph. The different initial particle positions are denoted by $P_0 \rightarrow P_2$. 
strongest drift of the particle to the left is encountered in the FLB-FLB simulation. In these cases, the diffusive effects of the fluid onto the particle have the highest level, with the fluctuations occurring on both grid levels. A similar behaviour, i.e. the drift to the left, is predicted by the BGK-FLB model as well though this drift is damped by the BGK operator, smoothing the flow field on the coarser grid level.

A second simulation set has been conducted where the oscillation frequency is chosen as $f = \frac{1}{10000}$. The particle movement is shown in the graph at the bottom of Fig. 16.6: identical drift effects as in the previous scenarios also show up in this case.

Concluding, the dynamic mesh refinement technique that has been presented in the subsection 10.5.2 has been validated in the context of moving sphere experiments. The results for position, velocity and force of the moving particle which is coupled bidirectionally to the flow problem were found to be in perfect agreement with non-adaptive simulations. Based on the dynamic refinement, a particle suspended in a nanopore has been simulated by means of purely fluctuating and hybrid BGK-FLB simulations. For an isolated particle in a fluid-filled box, the BGK-FLB approach was still able to capture the correct long-term diffusion of the particle. For the nanopore case, it could be shown that the Brownian fluctuations have a major impact on the particle movement. Both purely fluctuating and BGK-FLB simulation capture the same trend of the particle motion. However, in the BGK-FLB case, the missing thermal fluctuations on the coarse grid level damp the diffusive motion of the particle and thus yield a slighter drift.

Several steps can be taken from here to further improve the presented results. In the dynamically adaptive simulations, the fluctuations are incorporated on the coarse grid levels in the overlap regions of the adaptive grid. After prolongating the respective particle distributions to the fine grid, correlations are expected to occur since all $3D$ cells that are embedded in one coarse grid cell will transport the same particle distributions into the fine grid domain. One approach to reduce these correlations would be the incorporation of the fluctuating force terms directly on the fine grid cells and an extraction of the respective averaged forcing as a representation on the coarse grid level. Another issue remains which consists in the very small time steps. Considering the particle displacement from Fig. 16.6, one can observe that the time interval for $O(10)$ oscillations of the given scenario implies $O(10^6)$ time steps. This is acceptable for short-time two-dimensional simulations. However, for three-dimensional simulation runs or the investigation of the long-time behaviour of the particle, this poses a severe bottleneck. One improvement to reduce computational costs has been described by the dynamic mesh refinement technique in this section. Besides, massively parallel and highly optimised codes are required to overcome this issue. Another approach which also addresses the simulation of particle transport on longer time scales and which is orthogonal to the parallelisation and optimisation arguments may lie in the modification of the underlying particle transport models. A new hybrid model for particle transport has been described in the subsection 12.3.3. Results based on this approach are to be presented in the following section.

16.2 Long-Time Simulations: Lattice Boltzmann–Navier-Stokes Methods

Before applying the hybrid model from the subsection 12.3.3 to the particle transport problem, all of its components need to be validated.

First, the new optimisation-based approach which couples Navier-Stokes to Lattice Boltzmann (cf. Sec. 12.2) is investigated. For this purpose, a BGK-based Lattice Boltzmann simulation of plain channel flow is set up. Since the analytic solution—a parabolic velocity profile and a linear pressure drop—are known a priori, the optimisation approach for constructing the particle distribution functions can be applied at all outer boundaries of the channel. The mass, momentum and viscous stresses are imposed as boundary conditions onto the flow system. The approx. squared Knudsen-norm (cf. Sec. 12.2) is used as optimisation functional in this case. Since the channel flow only implies shear gradients, all aforementioned optimisation polynomials behave identically in this particular setup. In a first simulation suite, the channel flow is solved at constant Reynolds number, $Re = 0.075$. 

139
Table 16.2: Error analysis for the optimisation-based boundary conditions in plain channel flow. The first column depicts the number of grid points used in x- and y-direction, the second column shows the respective Mach number. The third column denotes the refinement factor that is the ratio of the mesh sizes between subsequent grid resolutions. The absolute discrete $L_2$-error is shown in the fourth column. The last column contains the error improvement factors, that is the factor between the current discrete $L_2$-error at a resolution of $N \times N$ points and the error at a coarser resolution of $N/2 \times N/2$ grid points.

<table>
<thead>
<tr>
<th>Grid points</th>
<th>Mach number</th>
<th>Refinement factor</th>
<th>Error</th>
<th>Improvement factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x4</td>
<td>8.66e-3</td>
<td>-</td>
<td>4.57e-5</td>
<td>-</td>
</tr>
<tr>
<td>8x8</td>
<td>4.33e-3</td>
<td>2.33</td>
<td>1.45e-5</td>
<td>3.16</td>
</tr>
<tr>
<td>16x16</td>
<td>2.17e-3</td>
<td>2.14</td>
<td>3.12e-6</td>
<td>4.64</td>
</tr>
<tr>
<td>32x32</td>
<td>1.08e-3</td>
<td>2.07</td>
<td>6.04e-7</td>
<td>5.16</td>
</tr>
<tr>
<td>64x64</td>
<td>5.41e-4</td>
<td>2.03</td>
<td>1.12e-7</td>
<td>5.38</td>
</tr>
<tr>
<td>128x128</td>
<td>2.71e-4</td>
<td>2.02</td>
<td>2.08e-8</td>
<td>5.39</td>
</tr>
</tbody>
</table>

Table 16.3: Error analysis for the optimisation-based boundary conditions in plain channel flow. In this scenario, the Mach number is held constant, $Ma = 1.73 \cdot 10^{-2}$, the average inlet velocity is fixed at $u_L = 0.01$ and the viscosity is chosen as $\nu_L = 0.33$.

<table>
<thead>
<tr>
<th>Grid points</th>
<th>Reynolds number</th>
<th>Refinement factor</th>
<th>Error</th>
<th>Improvement factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x4</td>
<td>9.00e-2</td>
<td>-</td>
<td>5.07e-4</td>
<td>-</td>
</tr>
<tr>
<td>8x8</td>
<td>2.10e-1</td>
<td>2.33</td>
<td>3.70e-4</td>
<td>1.37</td>
</tr>
<tr>
<td>16x16</td>
<td>4.50e-1</td>
<td>2.14</td>
<td>1.80e-4</td>
<td>2.06</td>
</tr>
<tr>
<td>32x32</td>
<td>9.30e-1</td>
<td>2.07</td>
<td>7.53e-5</td>
<td>2.39</td>
</tr>
<tr>
<td>64x64</td>
<td>1.89</td>
<td>2.03</td>
<td>2.95e-5</td>
<td>2.55</td>
</tr>
<tr>
<td>128x128</td>
<td>3.81</td>
<td>2.02</td>
<td>1.16e-5</td>
<td>2.55</td>
</tr>
</tbody>
</table>

on differently resolved grids. The Lattice Boltzmann cells are placed such that the outermost cell centres exactly coincide with the boundaries of the channel wall, inlet and outlet. The lattice viscosity is kept constant at $\nu_L = 0.2$ yielding a relaxation time $\tau = 1.1$. Doubling the number of Lattice Boltzmann cells and keeping the Reynolds number constant implies an approximate halving of the dimensionless inlet velocity and hence also a halving of the Mach number. After the steady-state solution is reached, the discrete $L_2$-error of the parabolic velocity profile is measured in the middle of the channel. The results are shown in Tab. 16.2. As expected for a second-order scheme, the error drops by a factor of $\sim 4$ when halving the mesh size. Analogous experiments are carried out at a constant Mach number $Ma = 1.73 \cdot 10^{-2}$ to check the influence of the compressibility errors from the previous experiments. For a constant Mach number, the compressibility effects are expected to remain of same order throughout all experiments. The Reynolds number increases by $\sim 2$ when doubling the number of grid points. The absolute error is consequently expected to be halved between subsequent grid resolutions. The results are summarised in Tab. 16.3 and agree very well with the theory.

Next, a two-way coupled Lattice Boltzmann–Navier-Stokes simulation for plain channel flow is considered which follows the description from the subsection 12.3.2: a two-dimensional grid consisting of $27 \times 27$ cells is set up for the Navier-Stokes solver with a Lattice Boltzmann domain embedded in the inner $9 \times 9$ Navier-Stokes cells. The LB domain has a resolution of $dx_{LB} = dx_{NS}/18$ which resembles a block size $N = 6$ per Peano cell and one level of refinement in the spacetree. The channel is initialised with zero velocity everywhere, and pressure Dirichlet conditions are prescribed at the in- and outlet. The convergence towards the parabolic profile is measured in the middle of the channel. Similar to the previous experiments, the discrete $L_2$-error is considered; in order to investigate the convergence towards
At data on the Navier-Stokes levels are corrected by the particle data from the Lattice Boltzmann simulation is carried out until it can fully develop. After releasing the particle at time \( t = 0 \), the particle-in-channel flow is expected in case of second-order boundary conditions applied for each solver. Due to the very fine LB grid, the convergence of the hybrid simulation should be faster than in the pure Navier-Stokes simulation. Still, the convergence is limited by the (first-order accurate) boundary conditions of the Navier-Stokes solver. The resulting convergence from the simulation is chosen as \( \Delta t = 0.07 \) and density \( \rho_{\text{stencil}} = 10 \rho_{\text{max}} \) is released in a channel of size \( 4.5 \times 1.0 \) at position \( (1.5, 0.5)^T \). The domain is discretised on four levels. On three coarse levels which consist of \( 9 \times 2, \ 27 \times 6 \) and \( 81 \times 18 \) cells, the Navier-Stokes equations are solved together with the Faxén post-processing step for the particle simulation. The Reynolds number for the Navier-Stokes solver is set to \( \text{Re} = 0.1 \), the time step of the Lattice Boltzmann region is set to \( B = 6 \) which yields a mesh size \( dx^{LB} = 3.08 \cdot 10^{-3} \). With the relaxation time chosen as \( \tau = 1.5 \), the time step of the Lattice Boltzmann simulation evolves at \( \Delta t^{LB} = 3.18 \cdot 10^{-7} \) which is two orders of magnitude smaller than the Navier-Stokes time step. We can hence track the movement of the particle on very short time scales and fine resolutions via explicit fluid-particle interaction. The simulation is started on the finest, that is on the Lattice Boltzmann, level. The particle is fixed from time \( t = 0 \) to \( t = 6.35 \cdot 10^{-3} \) so that the flow field in the Lattice Boltzmann region can fully develop. After releasing the particle at \( t = 6.35 \cdot 10^{-3} \), the Lattice Boltzmann simulation is carried out until \( t = 0.010 \). After each Navier-Stokes time step, the particle data on the Navier-Stokes levels are corrected by the particle data from the Lattice Boltzmann simulation. At \( t = 0.010 \), the Lattice Boltzmann domain is coarse-grained and the solution, the error is evaluated in each Navier-Stokes time step. The arising convergence graph is shown for both Lattice Boltzmann–Navier-Stokes and a pure Navier-Stokes simulation in Fig. 16.7. A slightly faster convergence of the hybrid simulation can be observed. Due to the very fine LB grid, the convergence of the hybrid simulation should be faster than in the pure Navier-Stokes simulation. Still, the convergence is limited by the (first-order accurate) boundary conditions of the Navier-Stokes solver. The resulting convergence from Fig. 16.7 is hence exactly in the range of what is expected for this scenario. The LB–NS interface conditions thus retain the convergence rate, and a second-order accurate solution is expected in case of second-order boundary conditions applied for each solver. With the Lattice Boltzmann–Navier-Stokes coupling validated, the hybrid particle transport approach from the subsection 12.3.3 is ready to be applied. A particle-in-channel flow scenario—similar to the scenario which has been used to validate the dynamic mesh refinement technique, cf. Sec. 16.1—is set up. It is visualised in Fig. 16.8. A spherical particle of diameter \( d_{\text{particle}} = 0.07 \) and density \( \rho_{\text{stencil}} = 10 \rho_{\text{max}} \) is released in a channel of size \( 4.5 \times 1.0 \) at position \( (1.5, 0.5)^T \). The domain is discretised on four levels. On three coarse levels which consist of \( 9 \times 2, \ 27 \times 6 \) and \( 81 \times 18 \) cells, the Navier-Stokes equations are solved together with the Faxén post-processing step for the particle simulation. The grid resolutions correspond to mesh sizes \( dx^{NS} = 0.500, \ 0.167 \) and \( 5.56 \cdot 10^{-2} \), respectively. The Reynolds number for the Navier-Stokes solver is set to \( \text{Re} = 0.1 \), the time step of the simulation is chosen as \( dt^{NS} = 3 \cdot 10^{-5} \). The fourth level of resolution corresponds to the next spacetime level. On this level, a block-structured Lattice Boltzmann region of size \( L_x \times L_y \) is embedded and centred around the particle. Different LB domain sizes have been investigated, \( L_x \times L_y \in \{ (0.375 \times 0.175), \ (0.75 \times 0.35), \ (1.5 \times 0.7), \ (2.0 \times 0.7) \} \). The block size of the Lattice Boltzmann simulation is set to \( B = 6 \) which yields a mesh size \( dx^{LB} = 3.08 \cdot 10^{-3} \). With the relaxation time chosen as \( \tau = 1.5 \), the time step of the Lattice Boltzmann simulation evolves at \( \Delta t^{LB} = 3.18 \cdot 10^{-7} \) which is two orders of magnitude smaller than the Navier-Stokes time step. We can hence track the movement of the particle on very short time scales and fine resolutions via explicit fluid-particle interaction. The simulation is started on the finest, that is on the Lattice Boltzmann, level. The particle is fixed from time \( t = 0 \) to \( t = 6.35 \cdot 10^{-3} \) so that the flow field in the Lattice Boltzmann region can fully develop. After releasing the particle at \( t = 6.35 \cdot 10^{-3} \), the Lattice Boltzmann simulation is carried out until \( t = 0.010 \). After each Navier-Stokes time step, the particle data on the Navier-Stokes levels are corrected by the particle data from the Lattice Boltzmann simulation. At \( t = 0.010 \), the Lattice Boltzmann domain is coarse-grained and the

Figure 16.7: Convergence towards the parabolic channel profile in a two-way coupled Lattice Boltzmann–Navier-Stokes simulation. The discrete \( L^2 \)-error of the velocity profile is plotted over time for the hybrid LB–NS approach and a pure Navier-Stokes simulation.
Figure 16.8: Hybrid Lattice Boltzmann–Navier-Stokes simulation for particle transport in a channel scenario. The simulation starts on the finest level and performs a two-way coupled fluid-particle simulation using the Lattice Boltzmann method. After predefined time intervals, the particle description is subsequently coarse-grained to the three coarse grid levels. On these levels, the particle trajectory is solved via the Faxén approach and the flow field is computed by a Navier-Stokes solver.

The simulation switches to the Navier-Stokes-Faxén approach on the finest Navier-Stokes grid level (level 2). The simulation is continued until \( t = 0.600 \). Then, the finest Navier-Stokes level is removed and the simulation continues on the next grid level (level 1) until \( t = 0.800 \). From \( t = 0.800 \), the fluid-particle simulation is carried out on the coarsest Navier-Stokes level (level 0). The x-coordinate of the particle and its velocity in x-direction are shown for different time intervals in Fig. 16.9. Both pure Navier-Stokes and three-level dynamically adaptive Lattice Boltzmann simulation results which are shown by the dark and light blue lines are in very good agreement. This illustrates the validity of both approaches: direct fluid-structure interaction between the particle and the flow field from the LB simulation and particle simulation via Navier-Stokes flow simulation and Faxén post-processing. Considering the long-time scale, that is the graphs of the first row in Fig. 16.9, it is observed that all particle positions match very well until the second Navier-Stokes coarsening from level 1 to level 0. Considering the particle velocities, jumps in the velocities occur when coarsening from the Navier-Stokes level 2 to level 1 (time \( t = 0.6 \)) and from level 1 to level 0 (time \( t = 0.8 \)). This is due to the very coarse representation of the flow field on these levels: the expression for the Faxén force (see Eq. (6.49)) involves both flow velocity and the Laplacian of the flow velocity which are naturally less accurate on the coarser grid levels. The shorter time scale of initial particle movement is shown in the mid row of Fig. 16.9. The transition from the Lattice Boltzmann to the Navier-Stokes simulation at \( t = 0.1 \) is smooth in terms of the particle movement. However, we can observe that the hybrid Lattice Boltzmann–Navier-Stokes simulations differ from the single-model solutions: in all hybrid cases, the particle moves faster than in the pure Navier-Stokes and Lattice Boltzmann simulation. Increasing the size of the embedded Lattice Boltzmann domain yields a better match. The same accuracy is observed for the LB domain sizes \( 1.5 \times 0.7 \) and \( 2.0 \times 0.7 \) indicating that the remaining error is only due to the finite domain size in y-direction. This error can be easily understood: since we use a one-way coupling of the flow field, the Lattice Boltzmann boundary is constantly initialised by the parabolic profile of the channel flow. However, due to the particle suspended in the flow, the profile of the fluid flow close to the particle is strongly affected. The heavy particle slows down the fluid near to it and thus significantly changes the overall flow profile on channel cross-sections close to the particle. This behaviour is not taken into consideration for the coupled approach and results in the slightly faster movement of the particle.
Reconsidering the illustration of the nanopore structures from Fig. 16.1, it can be observed that a huge number of nanopores is aligned in the respective material; the in-/outlet regions open up into a big reservoir. The hybrid Lattice Boltzmann–Navier-Stokes simulation shall in the following be used to simulate a small-sized particle which leaves a reservoir and enters one of the nanopores. The Navier-Stokes-Faxén approach is suited to simulate the particle moving in the large-scale reservoir. As soon as the particle approaches the nanopore, the simulation shall switch to the Lattice Boltzmann description. A two-dimensional scenario is set up: the computational domain is chosen to have a size $4 \times 3$. In the middle of the domain, three nanopores are embedded on top of each other, cf. Fig. 16.11. The domain is resolved on the Navier-Stokes level by $81 \times 54$ cells. A particle with diameter $d_{\text{particle}} = 0.08$ and density $\rho_{\text{particle}} = 5.0 \rho_{\text{fluid}}$ is suspended on the left side of the nanopores in the very middle between the reservoir channel walls. A constant parabolic profile is imposed on the left side to enforce the particle movement towards the pores. The Reynolds number is again chosen as $\text{Re} = 0.1$. The simulation starts by resolving the particle on the Navier-Stokes level using the Faxén approach. As soon as the distance of the particle to the nanopore structure...
drops below 0.04, the Lattice Boltzmann region is embedded. The mesh size of the Lattice Boltzmann simulation is chosen as in the previous particle experiments as $dx^{LB} = \frac{1}{18}dx^{NS}$.

In order to minimise disturbances due to the upper and lower boundaries of the LB region, the LB region covers two nanopores, cf. the second illustration in Fig. 16.11. After refining to Lattice Boltzmann, the LB distributions are initialised by the Navier-Stokes values. The LB system is equilibrated over a short time interval to remove any further disturbances that may arise from the different particle descriptions. During this period, the particle stays fixed at its location. Its boundaries are handled using the moving-wall boundary condition (see Sec. 5.5) together with the current particle velocity. After the equilibration phase, the particle is released and is explicitly resolved in the Lattice Boltzmann fluid.

The displacement of the particle and its velocity component in x-direction are shown in Fig. 16.10. The constant parabolic profile at the inlet region yields a continuous movement of the particle towards the right. Due to the narrowing at the left end of the nanopore, the particle is accelerated when entering this region. The pore chamber widens up afterwards yielding a decrease of the particle velocity in x-direction. Zooming into the graphs in Fig. 16.10, a tiny time interval can be noticed at $t \approx 0.7$ where both position and velocity remain constant. This time interval corresponds to the equilibration phase of the LB region during the transition from the Navier-Stokes to the Lattice Boltzmann solver. An enlarged representation of the time interval is shown in Fig. 16.12 for the displacement of the particle.

After equilibration, the particle moves forward again; the slope of its movement is retained which indicates the correctness of the presented approach for dynamically switching from the Navier-Stokes to the Lattice Boltzmann description.

Concluding, a new optimisation-based approach for coupling Navier-Stokes and Lattice Boltzmann solvers has been presented within this section. Compared to previous approaches by Latt et al. [110], more degrees of freedom exist in the optimisation-based approach to minimise the non-equilibrium parts of the particle distribution functions. These degrees of freedom arise from the choice of the function $g(f^{neq})$ to be optimised, cf. Sec. 12.2. In the presented studies, polynomials were chosen. The coefficients of the polynomials hence represent tunable parameters. Further investigations are required to develop strategies to tune these parameters and thus obtain optimal coupling descriptions for arbitrary physical problems. Within this context, more scenarios for hybrid Lattice Boltzmann–Navier-Stokes simulations need to be considered in future. After the validation in channel flows, the coupling methodology was applied to particle transport simulations in the low Reynolds number regime. The results suggest that good agreement can be obtained for the particle trajectories compared to pure Navier-Stokes and dynamically adaptive Lattice Boltzmann simulations. A major issue consists in the influence of the LB–NS boundaries onto the particle movement in the LB simulation: although the Navier-Stokes-Faxén approach does not explicitly resolve the particle in the fluid, the flow conditions of this approach are imposed onto the LB system. The modelling error in this step hence needs to be minimised. The particle-in-channel scenarios show that the boundaries which are aligned with the particle trajectory have the strongest influence. Therefore, these boundaries need to be either far away from...
Figure 16.11: LB–NS simulation for particle transport. A small-sized spherical particle enters a nanopore. Top: the particle is simulated by the Navier-Stokes-Faxén method and slowly approaches the pore. Mid: the switching from Navier-Stokes to Lattice Boltzmann is triggered. The grid is refined, and the Lattice Boltzmann region is initialised. Bottom: based on the Lattice Boltzmann description, the particle moves through the nanopore; a zoom into the Lattice Boltzmann grid close to the particle is provided at the upper right corner.
the particle or be generally negligible for the particle movement. The latter is for example
the case in the final scenario—transport of a particle from a big reservoir into a nanopore—
that has been presented in this section: due to the walls of the nanopores, the outer LB–NS
boundaries at the top and the bottom do not introduce errors to the particle movement. For
all test cases reported in this thesis, the fluid flow was highly laminar yielding steady flow
structures (except for the particle displacement and respective interactions with the fluid).
More research on consistently coupling Lattice Boltzmann and Navier-Stokes is required for
unsteady flows: temporal interpolation of the boundary conditions in the LB–NS overlap
region is required in these cases.

17 Molecular Dynamics–Lattice Boltzmann Coupling

Having discussed the coupling of Navier-Stokes and Lattice Boltzmann methods, hybrid
molecular dynamics–Lattice Boltzmann methods are the subject of this chapter. Therefore,
the different components of the macro-micro-coupling tool—introduced in Sec. 13.4—are
validated in Sec. 17.1. In the same section, the sequential and parallel performance of
coupled scenarios are discussed. One important aspect of molecular-continuum simulations
is given by the definition and implementation of open boundary conditions for molecular
dynamics. One respective model is based on the usage of radial distribution functions,
cf. Sec. 13.2. An extension of the original model implementation from [180] including an
analysis of its validity on the continuum and the molecular scale is presented in Sec. 17.2.
The chapter closes with Sec. 17.3 where all components for coupling Lattice Boltzmann and
molecular dynamics are combined into a hybrid scheme. Results for channel flow scenarios
are reported including first steps towards the LB–MD simulation of nanofilters.

17.1 MaMiCo: Validation and Performance

In order to validate the macro-micro-coupling tool, several test scenarios are considered in
the following. Most of the results have previously been published in [139]. For the tests, a
coupling of the built-in molecular dynamics solver (cf. Sec. 8.2) and a dummy continuum
solver is established. The dummy solver either sends a predefined mass or momentum
to all macroscopic cells that cover the molecular domain. First, the conservation of mass,
momentum and temperature by the controller mechanisms of the coupling tool are validated.
For this purpose, two test scenarios A and B are defined in Tab. 17.1. A domain of size
120 × 120 (2D) or 24 × 24 × 24 (3D) is considered. The Lennard-Jones parameters and the
mass of each particle are scaled to unity, σ = 1, ε = 1, m_p = 1. The temperature is set to
T = 6.1 in the two-dimensional cases and to T = 1.6 in the three-dimensional cases. The cut-
Table 17.1: Number density, momentum and temperature settings for test scenarios A and B in two and three dimensions [139].

<table>
<thead>
<tr>
<th>Test</th>
<th>Number density $n$</th>
<th>Momentum</th>
<th>Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test A 2D</td>
<td>$0.40 \rightarrow 0.78$</td>
<td>0.0</td>
<td>6.1</td>
</tr>
<tr>
<td>Test A 3D</td>
<td>$0.40 \rightarrow 0.78$</td>
<td>0.0</td>
<td>1.6</td>
</tr>
<tr>
<td>Test B 2D</td>
<td>0.80</td>
<td>11449 $\rightarrow$ 22898</td>
<td>6.1</td>
</tr>
<tr>
<td>Test B 3D</td>
<td>0.80</td>
<td>10648 $\rightarrow$ 21296</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Figure 17.1: Mass, momentum and temperature evolution for test scenario A [139]. The graphs represent the evolution of the three quantities over the 15000 time steps of mass insertion. Left: two-dimensional experiment. Right: three-dimensional experiment.

off radius is chosen as $r_c = 2^{1/6}$, allowing for purely repulsive molecular interactions. In test scenario A, the molecular system is equilibrated over 15000 time steps at a number density $n = 0.40$. Afterwards, the dummy solver sends enough mass to the molecular dynamics solver to nearly double the mass in the molecular domain. Over the next time interval of 15000 time steps, the mass from the dummy solver is inserted into the molecular system yielding an increase of the number density to $n = 0.78$. For mass insertion, the USHER scheme is applied. During the mass modification, momentum and temperature are locally conserved via the momentum and energy controllers. The resulting graphs for (global) mass, momentum and temperature are drawn for both the two- and the three-dimensional setup in Fig. 17.1. The mass insertion is more efficient at the beginning, since the USHER scheme can easily find valid molecule positions for low number densities. For increasing number densities, more steepest descent steps are required by USHER to insert a molecule. A slight drift in the momentum can be observed during the mass insertion. Considering the overall mass in the system, this drift amounts to a change of the average velocity in the total
Figure 17.2: Momentum and temperature evolution for the test scenario B [139]. The graphs represent the evolution of the two quantities over the 15000 time steps of momentum insertion. Left: two-dimensional experiment. Right: three-dimensional experiment.

The system of $O(10^{-3})$ which is considered to be negligible for most scenarios. The temperature decreases in the first steps approx. 0.8% (2D)/ 0.9% (3D) which is found to be due the strong changes in the energy landscape at the very beginning of mass insertion: at the same time, the thermostat is switched on in all macroscopic cells and mass insertion is started. Both mechanisms yield changes in the energy of the system. A slight drift during the first time steps is therefore expected. After these effects, temperature fluctuates around a mean value and remains constant.

The second test scenario B uses the same parameter and domain settings as test scenario A. This time, however, the number density is fixed at $n = 0.80$ and the average velocity of the molecular system is set to unity. This corresponds to an overall momentum of $j \approx 1.1 \cdot 10^4$. After 15000 time steps of equilibration, the momentum of the system is doubled via the dummy solver which sends the respective momentum contributions to the molecular dynamics simulation. The momentum is imposed to the molecular system using the additive momentum transfer strategy, cf. Sec. 13.2. The momentum transfer is carried out over 15000 time steps. The time evolution of momentum and temperature over the time interval of momentum insertion is shown in Fig. 17.2. Momentum increases linearly as dictated by the additive momentum transfer strategy. Temperature is conserved and naturally fluctuates around its mean value.

The runtimes for test scenarios A and B are depicted in Tab. 17.2. They are compared to similar molecular dynamics simulations. In case of test scenario A, both runtimes for molecular dynamics simulations at low and high number density are shown. In case of mass insertion, i.e. test scenario A, the simulation is significantly slower than both pure MD simulations. Comparing the performance to the averaged runtime $t_{\text{avg}}$ of the two reference MD simulations,

$$t_{\text{avg}} = \frac{t(\text{MD}(n = 0.40)) + t(\text{MD}(n = 0.78))}{2},$$

the runtime of the test scenario A evolves at

$$t_{2D}^A = 1.6 \cdot t_{\text{avg}}$$
$$t_{3D}^A = 2.1 \cdot t_{\text{avg}}$$

for the two- and the three-dimensional scenario.

The runtime for test scenario B is 14% (2D)/5% (3D) slower than the pure molecular dy-
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Runtime (s)</th>
<th>Timesteps/Particle insertion</th>
<th>Timesteps/Momentum insertion</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD ((n = 0.40)) (2D)</td>
<td>23.5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MD ((n = 0.78)) (2D)</td>
<td>46.6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test A (2D)</td>
<td>55.9</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>MD ((n = 0.80)) (2D)</td>
<td>48.0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test B ((n = 0.80)) (2D)</td>
<td>54.8</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>MD ((n = 0.40)) (3D)</td>
<td>92.1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MD ((n = 0.78)) (3D)</td>
<td>166.7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test A (3D)</td>
<td>268.4</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>MD ((n = 0.80)) (3D)</td>
<td>167.7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test B ((n = 0.80)) (3D)</td>
<td>175.2</td>
<td>0</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 17.2: Serial runtime (in seconds) for the test scenarios A and B compared to pure MD simulations [139].

Next, the parallel performance of the coupling tool is investigated. One of the challenging parts is the parallel USHER-based particle insertion; see Sec. 13.4 for its parallel extension which is incorporated in the macro-micro-coupling tool. Weak scaling experiments are conducted for the pure molecular dynamics simulation and a parallel particle insertion scheme. Two supercomputing platforms are used for these experiments:

1. **Huygens**\(^{13}\) is an IBM pSeries 575 machine. It consists of nine racks with each rack holding 12 compute nodes. Each compute node consists of 16 dual-core IBM Power6 processors which operate at 4.7 GHz.

2. **Shaheen**\(^{14}\) is a BlueGene/P architecture with 16 racks. Each rack contains 1024 quad-core PowerPC compute nodes where each core operates at 850 MHz.

For the pure built-in MD simulation of the coupling tool, two setups are investigated. In setup C, the MD domain size is chosen such that the execution of one MD time step in a serial simulation on Huygens is of the order \(O(0.1s)\). Setup D consists of a bigger domain. One time step on Huygens in the serial execution takes approx. 1s in this case. Since a huge number of time steps is required in molecular–continuum and purely molecular dynamics simulations, this choice is considered to be reasonable with respect to the later applications of the simulation code. The number density is chosen as \(n = 0.8\) (2D, according to the comparison of molecular dynamics with Lattice Boltzmann in [87]) and \(n = 0.6\) (3D, according to the hybrid molecular–continuum simulations in [43]). The scaling of the built-in molecular dynamics simulation is shown in Fig. 17.3; on Shaheen, only setup C is considered due to the significantly slower cores of this machine. The efficiency in the two-dimensional scenarios is found to be \(\geq 92\%\) on Huygens for \(\leq 1024\) cores and on Shaheen on \(\leq 4096\) cores. For the three-dimensional case, similar efficiencies are obtained on \(\leq 512\) cores. Performance drops are observed for higher core numbers on Shaheen.

Having evaluated the weak scaling of the molecular dynamics simulation, the weak scaling of parallel particle insertion tests is considered. A similar setup is chosen for the molecular dynamics domain with a number density \(n = 0.6\) (2D) and \(n = 0.5\) (3D). The molecular system is equilibrated for \(t_0 = 200\) time steps. Although this number of time steps is much too small to obtain a completely equilibrated molecular system, it is considered to be sufficient to form energy links between the molecules and thus construct an energy landscape for the USHER scheme. Over the next \(t_1 = 200\) time steps, mass is inserted in all macroscopic cells. This process is carried out analogously to the serial test scenario A: the dummy solver

---

\(^{13}\)http://sara.nl/systems/huygens/description

\(^{14}\)http://www.hpc.ku.edu/ documentation/user_guide/resources/shaheen/
is assumed to run on the same processes as the molecular dynamics simulation and sends mass to the MD system. The sent mass corresponds to a maximum change in the number density of $\Delta n = 0.2$. After the time interval $t_1$, the system is equilibrated again for another $t_2 = 200$ time steps. The efficiency of the weak scaling experiments is measured for the three time intervals $t_0$, $t_1$ and $t_2$ and is depicted in Fig. 17.4. The efficiency during the time interval $t_1$ where the particle insertion takes place is strictly above 93% for the considered core numbers on both platforms.

Finally, the parallel performance in a three-dimensional hybrid Lattice Boltzmann–molecular dynamics simulation shall be considered. For this purpose, the macro-micro-coupling tool is used to couple the built-in molecular dynamics simulation and the Lattice Boltzmann application of the Peano framework, cf. Chap. 10, for channel flow scenarios [136]. The molecular
dynamics simulation is embedded in the middle of the Lattice Boltzmann domain, cf. Fig. 17.5. An overlap layer of two Lattice Boltzmann cells is introduced where the velocity is mapped from Lattice Boltzmann to molecular dynamics. The molecules within this strip are relaxed towards the velocity received from the Lattice Boltzmann simulation following Eq. (13.11). In the inner part of the molecular dynamics domain, the average velocity is sampled for each macroscopic, that is Lattice Boltzmann, cell and sent to the Lattice Boltzmann simulation. In order to retain mass, the average density is evaluated in the outermost layer of macroscopic cells that is still part of the molecular dynamics region and the molecular system is pushed towards the reference density via molecule insertions and removals. Therefore, the difference between the average and the reference density is determined and the arising number of molecule insertions/ removals is triggered. The modification of mass is allowed in every time step of the molecular dynamics simulation. One coupling cycle is performed for this method which consists of two Lattice Boltzmann and 100 molecular dynamics time steps. This resembles a suitable choice for time-dependent scenarios; however, significantly more time steps are required by both solvers for steady-state based coupling strategies. The molecular dynamics solver is executed on all processes whereas the Lattice Boltzmann application is triggered only for a single process. The Lattice Boltzmann domain consists of $54 \times 54 \times 54$ cells. The (reference) number density in the MD simulation is set to $n = 0.6$ and the Lennard-Jones parameters are scaled to unity. The strong scaling for two scenarios is investigated: in scenario E, the mesh size of the macroscopic cells is chosen as $dx = 2.5$ and the number of molecules is set to $1.3 \cdot 10^5$. The scaling measurements are performed for 1, 8, 64 and 512 cores in this case. Scenario F uses macroscopic cells with $dx = 5.0$ and $1.0 \cdot 10^6$ molecules which corresponds to a molecular dynamics simulation which is eight times bigger than in scenario E. The number of macroscopic cells that cover the MD domains is held constant for both scenarios E and F to allow for a better comparison of both scenarios. The strong scaling is measured for scenario F on 1, 8, 64, 512 and 1728 cores. The simulations are conducted on Shaheen and Huygens. The arising speedup measurements together with the speedups of pure molecular dynamics simulations of same-sized MD systems are shown in Tab. 17.3 and 17.4. For small core counts, the speedups of the hybrid LB–MD simulations are comparable to the ones obtained in the pure MD simulations. For bigger core counts, the impact of the serial Lattice Boltzmann simulation becomes more and more dominant and reduces the overall speedup.

Within this section, the functionality of the macro-micro-coupling was validated and serial and parallel performance measurements were carried out. The serial performance of the USHER-based mass insertion is strongly dependent on the number density of the system. In the present test scenario A, the mass insertion tests took about twice the time of comparable
Table 17.3: Strong scaling for scenario E. The first column shows the number of processor cores followed by the speedup factors that were achieved on Shaheen and Huygens. For both platforms, the speedup in the hybrid LB–MD approach as well as the speedup in a comparable pure MD simulation are shown.

<table>
<thead>
<tr>
<th>Proc.</th>
<th>Shaheen</th>
<th>Huygens</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LB–MD</td>
<td>MD</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>8</td>
<td>6.6</td>
<td>6.8</td>
</tr>
<tr>
<td>64</td>
<td>36.0</td>
<td>44.0</td>
</tr>
<tr>
<td>512</td>
<td>105.4</td>
<td>206.6</td>
</tr>
</tbody>
</table>

Table 17.4: Strong scaling for scenario F.

non-coupled molecular dynamics simulations. Considering most algorithms for molecular–continuum simulations which apply the USHER scheme, two important aspects need to be mentioned. First, these algorithms do not intend to significantly change mass as it was done in the present cases. In test scenario A, mass was nearly doubled in the molecular dynamics region. In contrast, hybrid schemes, which for example combine molecular dynamics with incompressible or weakly compressible solvers, apply USHER to retain a constant density level. The arising number of insertions per MD time step is thus significantly smaller than in test scenario A. Second, mass modification schemes are typically only applied in an overlap or boundary region of the molecular domain. This reduces the order of mass insertions from \( O(N^D) \) to \( O(N^{D-1}) \) where \( N \) denotes the number of macroscopic cells in each dimension \( D \). Assume for example a molecular dynamics simulation which is performed at a number density \( 0.6 \) on a three-dimensional domain of size \( 24 \times 24 \times 24 \) and a boundary region which consists of one macroscopic cell. Each macroscopic cell shall have a size of \( dx = 2 \). The average number of molecules in each cell amounts to \( n \cdot dx^3 = 4.8 \approx 5 \). Further assume that maximum deviations in the number of molecules per cell occur of same order, that is \( 5 \pm 5 \) molecules per cell may be found. If the mass in the system is relaxed towards its reference density, this implies a maximum number of mass modifications

\[
N^{\text{mass mod.}} = (12^3 - 10^3) \cdot 5 \approx 3.6 \cdot 10^3. \quad (17.3)
\]

and a respective expected number of molecule insertions of \( 0.5N^{\text{mass mod.}} = 1.8 \cdot 10^3 \). The latter corresponds to the insertion of approx. 22% of the total number of molecules inside the molecular domain; this is a factor of five less than in test scenario A! For growing domain sizes, this behaviour further improves: for example, for a domain size of \( 50 \times 50 \times 50 \), the same computation yields an insertion of approx. 12% of the total number of molecules which results in a respective factor of nine. In this context, test scenario A can be considered as a worst case scenario. The cut-off radius in this test scenario was chosen very small. Increasing the cut-off radius will yield higher computational times for both the pure molecular dynamics simulations as well as the USHER-based particle insertion. Further tests are required in future to investigate the influence of the cut-off radius within this context.

For test scenario B, an overhead of approx. 10% was observed comparing the test scenario to non-coupled molecular dynamics simulations. The overhead results from three iterations over the molecules: in each time step, temperature needs to be evaluated (two iterations)
and re-set to the original value after the momentum insertion (one iteration). All these operations scale linearly in the number of molecules. With the cut-off radius \( r_c \) chosen very small, it is also the force computation that comes close to the (locally) linear scaling in this setup\(^\text{15}\). For bigger and thus physically more reasonable choices of \( r_c \), the influence of the momentum insertion is therefore expected to be significantly smaller. With this respect, the current test scenario B also represents a worst case scenario. Its performance results thus represent the lower bound with respect to the choice of the cut-off radius.

Still, some improvements in the serial performance of the coupling tool are possible such as the optimisation of the evaluation of average mass, momentum and temperature values in the macroscopic cells: at the moment, the evaluation of average mass, momentum and temperature are strictly separated within the momentum and the energy controller. This results in several iterations over the molecules. Although this clear separation facilitates the overall structure of the program, several evaluations may be merged into a single loop to increase performance. The respective improvements are subject of future work.

The capabilities of the coupling tool in parallel simulations were illustrated in parallel particle insertion tests and a hybrid simulation which coupled the built-in molecular dynamics code and the Lattice Boltzmann application of the Peano framework. In case of the parallel particle insertions, similar parallel efficiencies were obtained as in pure, that is non-coupled, molecular dynamics simulations. For the hybrid MD–LB scheme, similar speedups could be obtained as for the pure MD simulation for small core counts. In the presented scenarios, the LB solver was executed on a single core and hence represents a sequential part of the parallel simulation. In order to gain comparable speedups on bigger core counts, a parallel LB solver is required. First steps towards scalable LB simulations within the Peano framework have already been taken, cf. Chap. 14. The realisation of a fully parallel MD–LB simulation is therefore expected in near future. A validation of the hybrid simulation system for the considered channel flow scenarios has not been provided so far. It is subject of discussion in Sec. 17.3.

### 17.2 On RDF-Based Boundary Forcing

In this section, the issue of open boundaries in molecular dynamics simulations is addressed in more detail. A major issue is the construction of a consistent boundary force model which—to a certain extent—emulates a continuity in the potential energy landscape. No boundary force model is incorporated into the macro-micro-coupling tool yet. Although the methods and approaches listed in Sec. 13.2 have turned out to be suited for specific hybrid molecular–continuum simulations, some of these methods are still not completely understood. For example, it has been shown that the forcing model which is based on radial distribution functions (RDFs), cf. Sec. 13.2, outperforms other existing boundary models for fluids close to the supercritical state [180]. However, density oscillations have been reported for single-centred Lennard-Jones simulations in the liquid state [101]. The same behaviour has recently been observed for multi-centred Lennard-Jones simulations within the scope of a master’s thesis [102]. Hence, more research is required to evaluate the different methods in detail and subsequently incorporate them into the coupling software.

In the following, the RDF-based boundary forcing for single-centred molecules is studied. Its features and limitations are highlighted with respect to the approximation of the underlying integral equation

\[
\mathbf{F}_B(x) = n \int_{y \in \Omega_B} g(||y - x||) \mathbf{F}(y - x) dy. \tag{13.14}
\]

Reconsider the illustration of the integration volume \( \Omega_B \) in Fig. 13.4. For planar boundaries such as the one in this figure, Werder et al. use polar coordinates to evaluate the integral from Eq. (13.14). A best fit to the arising expression can be computed a priori. The computation of the boundary force during the coupled simulation reduces to the evaluation of the best fit-polynomial, solely depending on the distance \( r \) of the molecule from the open boundary.

\(^{15}\)The linked cells algorithm is already an algorithm of \( O(N) \). For a very small cut-off radius, the constant \( C \) in the underlying estimate becomes minimal.
Figure 17.6: Domain decomposition of a channel into molecular dynamics and continuum region. Due to global wall boundaries at the top and bottom, the open boundaries of the molecular dynamics simulation contain corners which are marked by red circles.

<table>
<thead>
<tr>
<th>cut-off radius $r_c$</th>
<th>2D</th>
<th>3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.12</td>
<td>1.12</td>
<td>3.0</td>
</tr>
<tr>
<td>1.5</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>3.0</td>
<td>3.0</td>
<td></td>
</tr>
<tr>
<td>4.0</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>5.0</td>
<td>5.0</td>
<td></td>
</tr>
<tr>
<td>6.0</td>
<td>6.0</td>
<td></td>
</tr>
<tr>
<td>discretisation width $dx$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td></td>
</tr>
</tbody>
</table>

Table 17.5: Cut-off radii and corresponding resolutions for the outer boundary discretisation for two- and three-dimensional scenarios.

This procedure is computationally cheap and consequently very efficient. However, it is not applicable for every scenario. For example, consider the domain decomposition for the channel from Fig. 17.6 into atomistic and continuum regions. Since open boundaries are only required at the left and right boundaries and with wall boundaries located at the top and bottom of the channel, the planar open boundaries degenerate to corners at the channel walls. A cheap integration using polar coordinates similar to the evaluation of the grey-coloured spherical cap in Fig. 13.4 is thus not possible in these regions. Another example is given by molecular dynamics simulations of more complex substances which are modelled by multiple interaction sites per molecule [102]. In this case, the symmetry of the molecules towards the boundary is broken, and more flexible integration methods are required.

One of the simplest approaches to numerical quadrature is given by the midpoint rule (see Sec. 13.2) and is used in the following. This scheme has been incorporated into the built-in molecular dynamics simulation of the macro-micro-coupling tool. Studies have been conducted in two- and three-dimensional molecular dynamics simulations to validate the integration scheme and investigate the influence of the discretisation cell size $dx := dx/M$ where $dx$ is the size of one linked cell and $M$ is the number of discretisation cells per linked cell, cf. Sec. 13.2. In the two-dimensional case, the domain size of the molecular dynamics simulation is chosen as $144 \times 144$, and the number density is set to $n = 0.594 \approx 0.6$ resembling a total number of 12321 molecules. The Lennard-Jones parameters are set to unity, $\epsilon = 1$, $\sigma = 1$, and temperature is scaled to $T \approx 6.4$. Different cut-off radii are investigated, $r_c \in \{1.12, 1.5, 2.0, 3.0, 4.0, 5.0, 6.0\}$. For each cut-off radius, different resolutions for the underlying quadrature are applied, see Tab. 17.5. The three-dimensional setup consists of 16250 molecules in a cubic domain of size $30 \times 30 \times 30$ yielding a number density $n = 0.602 \approx 0.6$. The Lennard-Jones parameters are also scaled to one. The temperature is set to 1.8 so that the molecular description matches the one from Werder et al. [180]. Two values for the cut-off radius are used, $r_c \in \{1.12, 3.0\}$. The resolutions for the quadrature of the RDF integral expression are also listed in Tab. 17.5.
Table 17.6: Relative error (in %) for the RDF-based boundary force compared to force contributions in periodic MD simulations.

<table>
<thead>
<tr>
<th>discretisation width $dx$</th>
<th>cut-off radius $r_c$, 2D</th>
<th>$r_c$, 3D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.12</td>
<td>1.5</td>
</tr>
<tr>
<td>0.1</td>
<td>0.15</td>
<td>0.44</td>
</tr>
<tr>
<td>0.2</td>
<td>0.01</td>
<td>–</td>
</tr>
<tr>
<td>0.3</td>
<td>–</td>
<td>0.84</td>
</tr>
<tr>
<td>0.4</td>
<td>3.23</td>
<td>–</td>
</tr>
<tr>
<td>0.5</td>
<td>–</td>
<td>7.07</td>
</tr>
<tr>
<td>0.6</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

In a first test suite, after equilibrating the MD systems, 100 000 time steps are computed using periodic boundary conditions. For all molecules that are inside the computational domain and that are located in the first linked cell layer next to the left boundary, the total RDF-based boundary force is evaluated. For this purpose, the RDF-based forcing term which would act in a coupled simulation onto these molecules from the left boundary region is evaluated for each respective molecule. The arising forcing terms are summed up and averaged over time. Note that this force is not applied to the molecules; it is solely evaluated based on the movement of the molecules within this periodic setup. Besides, the force that acts onto the molecules across the left boundary due to the periodic boundary conditions is evaluated, and its average value is determined. The relative error of the average forces

$$
\epsilon_{rel} := \frac{|F_{\text{RDF}} - F_{\text{Periodic}}|}{F_{\text{Periodic}}} \quad \text{(17.4)}
$$

is measured where $F_{\text{RDF}}$ denotes the average RDF-based boundary force and $F_{\text{Periodic}}$ represents the force from the periodic boundaries.

In case that the local RDF-based boundary force exactly represents the force that arises from the periodic boundary conditions in every time step, the molecules close the boundary would follow exactly the same trajectory. The error between the periodic and the RDF-based forces would vanish in this case. Since the random molecular information is not available in coupled molecular–continuum simulations, a perfect match of the trajectories is however not expected. From the continuum point of view, a “sufficient” match may consist in a force over the molecular–continuum boundary which yields similar averages and variances as found from periodic MD simulations. A correct average force is further consistent with the requirement of retaining the correct mean pressure [180]. The relative errors for the different scenarios are shown in Tab. 17.6. In most scenarios, that is in all scenarios except for the cut-off radius $r_c = 5.0$ (2D) and $r_c = 3.0$ (3D), a resolution of $dx \leq 0.4$ has been found sufficient to obtain errors in the average boundary force of less than 5%. A correct mean boundary force is thus imposed via the RDF-based boundary force approach.

Next, the influence of the open boundary model with respect to the macroscopic quantities is investigated. For this purpose, the lower and upper boundary of the two- and three-dimensional simulations are turned from periodic into open boundaries. Molecules that are close to an open boundary now feel the RDF-based boundary force. If a molecule crosses the boundary, the normal components of its translational velocity are inverted and its position is updated following a reflection at the boundary plane (specular reflection). Since a fluid at rest is considered in the present study, the considerations are restricted to number density measurements. The profile of the number density is determined on lines that are either parallel or orthogonal to the open boundary. Each line is drawn through the midpoints of linked cells. The sampling of the number density is accomplished within these linked cells over 200 000 time steps. Before sampling, the system is equilibrated for 15 000 time steps, starting from the equilibrated purely periodic system. Different exemplary profile plots are shown in Fig. 17.7 and 17.8: the density profile naturally fluctuates throughout the domain.
in all simulations. On the linked cell level, the fluctuations close to the open boundary are of same order as in the periodic simulations.

As a last point, the molecular structure close to the boundary is further investigated. Therefore, the RDF is determined in the open boundary simulations and compared to the RDF from purely periodic simulations. In order to capture the influence of the open boundary, the RDF should not be sampled within the whole simulation domain, but inside a region which is as close to the open boundary as possible. Measuring the RDF directly on the boundary is difficult since no molecules can be found on the other side of the open boundary. Here, the sampling region is chosen to be the second layer of linked cells inside the computational domain, that is the x-stripe (2D)/ x-y-plane (3D) of linked cells which is exactly one linked cell width away from the open boundary. The arising radial distribution functions and the radial distribution functions sampled from periodic MD simulations are shown in Fig. 17.9. The cut-off radii were chosen as $r_c = 1.12$ and $r_c = 3.0$. All distributions—from periodic as well as from open boundary simulations—are nearly identical. Only for the cut-off radius $r_c = 1.12$, the initial peak is slightly underestimated in both two- and three-dimensional simulations.

Concluding this section, the RDF-based boundary force developed by Werder et al. [180] has been successfully incorporated into the built-in molecular dynamics simulation of the macro-micro-coupling tool. A simple numerical quadrature has been applied to allow for the application of the RDF-based boundary forcing for arbitrary boundary shapes. The results suggest that this scheme provides a sufficient order of accuracy for discretisation widths $dx \leq 0.4$ in the presented studies. Although this cell size is quite large compared to the characteristic form of the radial distribution functions, the computational costs are still high in this case. For example, for a small cut-off radius $r_c = 1.2$, a number density $n = 0.6$ and a discretisation width $dx = 0.4$, nine evaluations of the intermolecular force model are required in case of a 2D simulation, although periodic MD simulations would only require one force evaluation. In the 3D case, it is even 27 force evaluations! A mixed approach which only involves this computationally expensive treatment at corners or other complex open boundaries and uses the cheap polar coordinate-based integration from [180] everywhere else is thus highly favourable. The boundary model was validated by measuring the average force across the open boundary. The measured force was found to be in good agreement with the force from periodic MD simulations in both 2D and 3D simulations. In addition, the number density was measured on different cuts through the simulation domain. On the length scale of the linked cell discretisation, the resulting profiles showed the same level of thermal fluctuation in open boundary and periodic simulations. In order to check for consistency with respect to the molecular structure of the fluid, the radial distribution function was sampled close to the open boundary region, that is in a distance of one linked

Figure 17.7: Density profiles for two- and three-dimensional scenarios with the RDF-based boundary force applied at the lower and upper boundary of the MD domain. The cut-off radius is chosen as $r_c = 1.12$ and the linked cell size is set to $dx^{lc} = 1.2$. The profile is plotted over a line which is orthogonal to the open boundary planes. The parameter $M$ denotes the number of discretisation cells per linked cell and spatial direction. Left: 2D. Right: 3D.
Figure 17.8: Density profiles for two- and three-dimensional scenarios with the RDF-based boundary force applied at the lower and upper boundary of the MD domain. The cut-off radius is chosen as $r_c = 1.12$ and the linked cell size is set to $d_{lc} = 1.2$. The profile is plotted over different lines which are parallel to the open boundary planes. The distance to the open boundary is given in the legend. The parameter $M$ denotes the number of discretisation cells per linked cell and spatial direction. Left: 2D. Right: 3D.
cell width. The arising RDFs were nearly identical to the ones from periodic simulations. Still, measurements closer to the open boundary would be highly desirable. This is particularly important if the overlap layer in a molecular–continuum simulation shall be reduced to a minimum and thus in the optimal case to zero width: in this case, the fluxes that are exchanged over the surface between the continuum and MD simulation need to be as consistent as possible on both sides, and any induced modelling error may have even more severe consequences on the overall stability and accuracy of the coupled simulation than in other hybrid approaches. For Schwarz-like coupling approaches such as the ones from [43, 180], the overlap layer already acts as a “buffer” for respective modelling errors. Based on the experiments within this section, the disturbances very close to the open MD boundary are therefore expected to play a minor role for these schemes.

17.3 Towards the Molecular–Continuum Simulation of Nanofilters

After the short excursion to boundary forcing, molecular–continuum simulations of channel-like flow scenarios are considered in the following. Although channel scenarios are described by a very simple geometrical setup, they represent a sophisticated validation scenario for the hybrid schemes. First, similar to the rather technical test scenarios from Sec. 17.1, the correct cellwise transfer of quantities between the mesh-based continuum and the MD solver can be evaluated. Second, with a parabolic profile expected as average solution in case of non-rarefied flow simulations, the coupling scheme can be tested for second-order accuracy: the velocity profile should thus be captured correctly by the hybrid method. Third, placing the MD region for example in the middle of the channel, cf. Fig. 17.5, quantity transfer in both tangential and normal direction to the molecular–continuum interface is required. Both directions can thus be investigated using the channel scenario.

Particular channel-like scenarios are given by nanofilters, see Fig. 17.10. These filters consist of very fine-grained porous structures or membranes which remove ions or other pollutants from a solvent. Nanofiltration can be used for example to clean water [153]. Considering the
big amounts of drinking water required all over the world, highly efficient filters are of major importance, removing as much pollutants as possible at a maximum flow rate. Numerical simulation represents a useful tool for respective design studies: exemplary studies using MD simulations have been conducted in [57] where a particular membrane design based on Carbon nanotubes is shown to allow for significantly higher water fluxes than commercial nanofiltration membranes.

The simulation of the flow in nanofilters can be very time-consuming: the filter needs to be modelled on the molecular scale and thus requires a computationally intensive molecular dynamics simulation. The flow field far away from the porous structure, however, does not contain any molecular obstacles. Hybrid simulations can thus be employed as illustrated in Fig. 17.10: the in- and outlet region are resolved by a continuum or mesoscopic method, for example a Lattice Boltzmann solver, whereas a microscopic method like molecular dynamics is used close to the filter. A respective hybrid continuum–DSMC approach for gas flows has been presented in [3]. In the following, steps towards the simulation of nanofilters for liquids are taken.

A hybrid Lattice Boltzmann–molecular dynamics scheme has been implemented based on the approach presented by Dupuis et al. [43] as reviewed in Sec. 13.3. It is established using the macro-micro-coupling tool which connects the built-in molecular dynamics simulation of the coupling tool and the Lattice Boltzmann application of the Peano framework (cf. Sec. 17.1).

Some modifications to the original algorithm from [43] have been incorporated. Since the porous structures of the filter are strictly three-dimensional, cf. Fig. 17.10, the overall simulation coupling is established in 3D, that is using the 3D built-in molecular dynamics solver of the macro-micro-coupling tool and the 3D Lattice Boltzmann simulation of the Peano framework. In the original work by Dupuis et al., a 2D Lattice Boltzmann solver has been used. Sampling in the MD simulation could therefore be carried out over the third spatial dimension in the MD simulation which strongly reduces fluctuations in the averaged quantities. Besides, the size of the Lattice Boltzmann cells is chosen to be comparable to the molecular diameter [43]. Despite the questionability of such small cells on LB side (see Sec. 13.1), the reduction of fluctuations during sampling within such small volumes would require extremely long sampling times in case of a fully three-dimensional setup. Hence, bigger macroscopic and respective LB cells are used in the present approach.

Since the cells are chosen significantly bigger than in the original studies by Dupuis et al., the cellwise velocity relaxation process requires additional modifications. If a molecule moves from one macroscopic cell into another one, the target velocity in the relaxation process changes for this molecule. For very small sampling cells such as the ones used in [43], the transition for the molecule is still very smooth. The small sampling cells represent rather point- than cellwise data in this case which facilitates the pointwise relaxation process. Besides, only very small changes in the velocities occur between neighbouring cells. Hence, a smooth transition is expected for the molecules when leaving one and entering a new sampling cell. Using bigger sampling cells, the velocity value sampled in a single cell cannot
Figure 17.11: Velocity relaxation schemes in hybrid MD–LB simulations. The velocity of the red-coloured molecule is pushed towards a target velocity which is interpolated from the velocity values of the neighbouring macroscopic cells. The velocity values of the macroscopic cells are shown by green arrows. Left: $d$-linear interpolation. Right: second-order interpolation.

be interpreted as a pointwise average value anymore. In order to provide a smooth velocity profile in the relaxation procedure, the target velocity needs to be interpolated at the position of each individual molecule. Two interpolation schemes are investigated:

- **$d$-linear**: a $d$-linear interpolation of the channel flow velocity is used to determine the target velocity for a specific molecule. The interpolation is carried out between the midpoints of the macroscopic cells as shown in Fig. 17.11 on the left. The overlap layer is thus required to have a thickness of at least two macroscopic cells.

- **Second-order**: the shape of the velocity profile is determined by second-order interpolation as depicted in Fig. 17.11 on the right. The triangle (2D)/ tetrahedron (3D) which contains the specific molecule and is spanned by macroscopic cell midpoints is determined. The interpolation is subsequently carried out. A thickness of at least three macroscopic cells is required for the overlap layer in this case.

In order to impose the sampled average MD velocity in the LB simulation, a similar forcing technique as the one presented in [43] is used. An additional relaxation parameter is introduced on the LB side to further smoothen the state transition. After the collision step, a forcing term is incorporated into the post-collision Lattice Boltzmann state as follows:

$$f_i(x + c_i dt, t + dt) := f_i^*(x, t) + \frac{w_i}{c_s^2} \rho(x, t) \lambda (c_i (u^{MD} - u^{LB}))$$  \hfill (17.5)

where $u^{LB}$ represents the current LB velocity and $u^{MD}$ the velocity from the MD simulation that should be imposed onto the LB system. The relaxation parameter $\lambda$ is chosen from $(0, 1]$ where $\lambda = 1$ corresponds to the general methodology from [43].

The modified scheme is validated in three-dimensional channel flow. The channel has a dimensionless size$^{30}$ of $135 \times 135 \times 135$, with the MD simulation placed in the very middle of the channel and occupying a space of $60 \times 60 \times 60$. The domain is covered by $54 \times 54 \times 54$ LB cells which results in a mesh size $dx^{LB} = 2.5$ for the Lattice Boltzmann—and thus the macroscopic—cells. The parameters in the MD simulation are chosen according to the settings in [43], that is $m_p = 1.0$, $\sigma = 1.0$, $\epsilon = 1.0$, $T = 1.8$, $\nu = 1.4$ and $n = 0.6$, which corresponds to liquid argon. This results in approx. 130 000 molecules in the MD simulation. Periodic conditions are applied at all boundaries of the MD region. Since the main flow direction in the channel is from left to right, the stresses at the front, back, top and

---

$^{30}$ The dimensionless quantities refer to the scaling in the MD simulation.
Figure 17.12: Profiles in a hybrid MD–LB simulation using different interpolation schemes for the velocity relaxation on MD side. Left: $d$-linear interpolation. Right: second-order interpolation.

Bottom boundary planes of the MD simulation cannot be captured correctly in the MD simulation with periodic boundaries. For this reason, a boundary strip of $0.5 d x^{LB}$ is introduced in the MD simulation. No measurements or velocity relaxation are carried out in this part to let the molecules adapt to the new flow conditions. Within the subsequent boundary strip of size $d x^{LB}$, the velocity of the channel flow is imposed onto the molecular system. The interpolated velocity relaxation procedure, cf. Fig. 17.11, is applied in this strip. After equilibrating the MD system at zero velocity and equilibrating the channel on the Lattice Boltzmann level until the flow profile has reached steady state, one coupling cycle is solved. The cycle consists of 2000 LB time steps ($d t^{LB} = 0.22$) and 300 000 MD time steps ($d t^{MD} = 0.002$): the MD system is equilibrated under the new flow conditions over 240 000 time steps, and the sampling of average velocities is carried out over the following 60 000 time steps. The average inlet velocity is chosen as $u = 0.45$. The relaxation parameter on LB side is set to $\lambda = 0.1$ whereas a relaxation parameter of 0.05 is chosen for the MD simulation. The arising normalised velocity profiles at a cross section through the MD–LB domain are shown for both interpolation schemes in Fig. 17.12. The $d$-linear interpolation scheme yields a flattening of the profile whereas an accurate profile curvature is recovered in case of the second-order interpolation.

Stepping towards realistic nanofilter scenarios, a geometrical setup of a filter device is created, cf. Fig. 17.13: two reservoir chambers are modelled. The chambers are connected by a small duct which is to contain a filter membrane. The (dimensionless) size of each chamber is set to $95 \times 135 \times 135$, the size of the duct is chosen as $80 \times 55 \times 55$. The size of the Lattice Boltzmann cells is set to $d x^{LB} = 2.5$ as in the channel scenario. Besides, the same equilibrated MD system as in the previous channel experiments is used in this simulation. It is embedded in the center of the small-sized duct. With the MD region spanning a size of $60 \times 60 \times 60$, layers of size $60 \times 2.5 \times 2.5$ would be placed inside the duct walls. The molecules inside these layers are frozen and stay fixed in space throughout the simulation to model rigid walls on the molecular scale. Besides, a membrane geometry is defined as shown in the lower right part of Fig. 17.13. All molecules inside this grid-like geometry are also frozen and thus model the filter membrane. The membrane is only “visible” on the molecular scale; it is incorporated on the LB scale only via coarse-graining of the average molecular velocities. In order to allow molecules to leave and enter the MD region on the left (inlet) and right (outlet) side, periodic conditions are applied together with the buffer region of thickness $0.5 d x^{LB}$. Due to the filter-like obstacle, density variations may occur and may not be consistent with periodicity. Therefore, the mass is sampled in the outermost macroscopic cell layer on the left and right side of the MD domain over time intervals of 10 000 MD time steps. The difference between the reference mass $m^{ref} = m_p \cdot n \cdot d x^{LB}$ and this sampled mass $m^{sampled}$ is imposed over the subsequent 10 000 time steps in the respective macroscopic cell via USHER-based particle insertion and randomised particle removal. This
Figure 17.13: MD–LB simulation of flow through a nanofilter. Top: visualisation of the MD region and transition from the Lattice Boltzmann flow field—shown by coloured arrows—to the molecular flow field. Blue molecules are freely moving whereas red molecules are fixed in space. The latter are used to model rigid walls and the filter membrane. Lower left: side view of the geometrical setup. The setup consists of two large-scale reservoirs which are connected by a small duct with the filter membrane. The molecular walls and membrane are shown by red molecules. Lower right: oblique view of the geometrical setup.

methodology retains a constant density at the interfaces close to the weakly compressible LB simulation. The velocity relaxation is carried out on the left and right side of the MD simulation following the channel flow description; the second-order interpolation technique is used in this case.

The LB simulation is initially equilibrated over 80 000 time steps to yield a steady solution of the flow in the reservoirs and the connecting duct, neglecting the influence of the filter membrane. Then, one coupling cycle of the hybrid LB–MD scenario is simulated. With the LB–MD boundaries placed at a sufficient distance from the filter membrane, the coarse-grained results from the MD simulation provide a first estimate for the LB representation of the filter structure after this first coupling cycle. In order to account for the new molecular boundaries in form of frozen molecules, the MD simulation is equilibrated for 800 000 time steps which is significantly longer than in the pure channel scenario. The flow field data are sampled over 200 000 time steps and sent back to the LB solver. Flow field data of
the simulation are shown in Fig. 17.14: the velocity profiles on MD and LB scale show the correct qualitative behaviour, with high velocities in the center of the filter and vanishing velocities at the filter surface and the channel walls.

Concluding, a coupled simulation software has been developed in this section, integrating all components of the macro-micro-coupling tool, the Peano-based Lattice Boltzmann application and the built-in molecular dynamics simulation of the coupling tool. The hybrid scheme was validated in three-dimensional channel flow simulations. Besides, first steps towards the simulation of more complex scenarios have been taken. In this context, qualitative results for flows in nanofilters have been presented. A detailed validation and quantitative comparison of the simulation results with pure MD simulations of the overall computational domain are subject of future work.
In this thesis, new algorithms and software concepts have been presented for flow simulations on continuum, mesoscopic and molecular scales. Three different types of solvers have been applied: Navier-Stokes solvers for the continuum flow description, Lattice Boltzmann methods for the mesoscopic scale and molecular dynamics for the molecular scale. A particular focus was put on the development of the mesoscopic flow simulation software as well as on the coupling of the different scale descriptions. Addressing the latter, a new coupling strategy for hybrid Lattice Boltzmann–Navier-Stokes simulations has been presented. Besides, in order to facilitate the development of (massively parallel) molecular–continuum simulations, the macro-micro-coupling tool was designed and used in different molecular dynamics–Lattice Boltzmann scenarios.

Spatially Adaptive Lattice Boltzmann Simulations in Peano

The mesoscopic and continuum simulations have been carried out using the Peano framework. In order to extend the application range of the framework to the mesoscopic scale, a spatially adaptive Lattice Boltzmann implementation was established. Besides the standard Lattice Boltzmann collision operators and boundary conditions, enhancements for simulations close to the microscopic scale have been included which allow for fluctuating and rarefied gas simulations. The integration of these two extensions into the existing adaptive LB formulation was rather straightforward due to the locality of the respective algorithmic steps. Several steps can be taken to further improve the simulation software. From the author’s point of view, two particular extensions are of major importance. The first one is given by the extension of the adaptive LB scheme to second order. This requires interpolation techniques and thus non-local operations on the space tree grid of the Peano framework. These operations are supported between vertices which also represent the main data structures for the block-structured LB grid so that no particular difficulties are expected with respect to this step. Second, an efficient distributed memory parallelisation needs to be established for the adaptive LB implementation. With Peano providing the callback structures for this step as well, this extension does not seem difficult to be incorporated at first sight. However, due to the block grid management system which is provided to the LB application from “outside” the Peano kernel, several communication steps need to be implemented by the developer. The callbacks may be reused to trigger communication, the MPI implementation, however, is not completely hidden from this kind of application anymore. First steps towards improving the kernel with respect to external memory management are already taken in Peano, V. 3, which is just about to evolve. Besides, the experiments on the regular grid implementation revealed that an efficient implementation is possible if the communication of the big data sets, that is the particle distribution functions, can be reduced. Still, a certain overhead arises in Peano, V. 2, from the communication of the pure vertex records in each time step. For the space tree management of the Peano kernel, this communication is important to keep the LB block structure consist over all processes and grid iterations. Reducing this overhead further by only communicating the vertex records when changes of the grid topology occur...
hence represents an interesting aspect, not only for the Lattice Boltzmann application, but also for other applications which do not require synchronisation in each grid iteration. Besides the static adaptivity, a dynamic mesh refinement technique was developed and validated for flow simulations with moving geometries. The results obtained in these simulations agreed very well with corresponding results from simulations using non-adaptive grids. However, the topology changes due to a moving sphere are—despite the generality of this scenario with respect to the spacetree grid structure—rather moderate. The application of the dynamic mesh refinement technique in more complex flow scenarios would hence be very interesting. One example comprises the simulation of free surface flows in which the simulated liquid has a freely moving boundary (liquid-gas interface), cf. Fig. 18.1. Tracking the liquid-gas interface via resolving this boundary region on the finest grid level thus represents a challenging scenario for the dynamic mesh refinement technique and may reveal further information with respect to accuracy and performance.

**Optimisation-Based Coupling of Lattice Boltzmann and Navier-Stokes Solvers**

Besides the spatially adaptive Lattice Boltzmann solver in Peano, a new optimisation-based approach was developed to consistently couple mesoscopic Lattice Boltzmann and continuum Navier-Stokes flow simulations. The scheme was successfully validated in channel flow scenarios and employed to particle transport problems. It was further shown that in case of vanishing bulk stresses, the presented scheme and a previously developed approach [110] are identical. With the new scheme providing tunable degrees of freedom (in terms of the function $g(f^{\text{neq}})$ which should be minimised), more investigations are required to exploit this feature. Since the non-equilibrium parts $f^{\text{neq}}$ form the viscous stresses, scenarios with strong gradients of the form $\partial_{x_\alpha} u_\alpha$ may represent interesting setups in this context. Different spatial and temporal resolutions for the Lattice Boltzmann and Navier-Stokes solvers were used throughout the scenarios within this thesis\(^1\). This indicates the applicability of the hybrid method on non-uniform grids as well as the possibility to resolve small scale effects such as Brownian fluctuations on finer grids. Until now, the coupling was established for steady state scenarios with respect to the flow field. The validation of the coupling for unsteady flow scenarios hence still needs to be car-

\(^1\) Non-adaptive Cartesian grids of some resolution have been employed in the original publication on the optimisation-based approach [134].
ried out. For unsteady flows, however, more work is required to correctly model the LB–NS interface conditions. For example, temporal interpolation needs to be applied if different time step sizes are used for both solvers.

Besides the validation aspects and further extensions of the hybrid scheme, the simulation of more complex systems via the coupled Lattice Boltzmann–Navier-Stokes approach needs to be established to point out its efficiency, exploiting both solver strategies as much as possible. Examples comprise laminar flows in domains which consist of large-scale regular regions on the one hand and small-scale porous structures on the other hand; the application of the Lattice Boltzmann method is advantageous for the porous medium whereas (implicit) Navier-Stokes solvers should be superior when handling the regular regions.

The Macro-Micro-Coupling Tool for Hybrid Molecular–Continuum Simulations

In order to simulate large systems close to the molecular regime, the macro-micro-coupling tool was developed according to principles of software engineering (modularity, reusability, extensibility) and high-performance computing (MPI-based parallelisation of the coupling), cf. Fig. 2.3 from the introductory part on software requirements. The tool allows to establish the scale transition in hybrid molecular–continuum simulations. The functional components of the tool have been validated in various test cases. With respect to the parallelisation of the different coupling steps, an existing parallel algorithm of the USHER scheme for particle insertion has been further extended to reduce the amount of energy evaluations and also allow for shared memory parallelisation. The algorithm was incorporated into the coupling tool and was successfully applied in the parallel simulations. Within the scope of this thesis, a coupling of a relatively simple MD solver with the Lattice Boltzmann application of the Peano framework was established and applied to different channel-like scenarios, indicating the suitability of the overall coupling tool design on the one hand and showing the successful integration of the different coupling steps for a particular coupling scheme on the other hand.

Two major directions for future development can be defined to further enhance the coupling tool. The first direction is given by the incorporation of more functionality into the coupling tool: with a higher level of functionality available, a facilitated setup of new coupling schemes is accomplished which further indicates the general applicability of the tool. Examples for respective functional extensions comprise different velocity and momentum transfer operations or new mass transfer schemes such as the USHER scheme for multi-centred molecule types [32]. Another very important ingredient of many hybrid molecular–continuum simulations is the treatment of open boundaries in the molecular system. One particular method to handle open boundaries is given by a combination of the RDF-based continuum forcing with reflecting boundary conditions for the molecules and has been studied in this thesis, cf. Sec. 17.2. Due to the limitations of this particular scheme and the variety of different boundary models that has been reported so far, a general interface structure to incorporate this functionality into the coupling software has not been identified yet; more work on open boundary models and their integration into the coupling tool will follow.

The second direction aims at further pointing out the reusability of the coupling tool with respect to coupling different continuum and molecular dynamics solvers. So far, the coupling was established for one Lattice Boltzmann solver and a dummy continuum solver (for the validation cases) as well as one molecular dynamics simulation. In this context, the interface definitions which evolved from requirements of the software analysis in hybrid schemes (cf. Sec. 13.3) have shown to be flexible enough in all tests. Coupling different pieces of software, however, is required to further prove the general applicability of the current interface definitions.

Scientific Impact

Concluding, different research aspects of multiscale flow simulation have been addressed. First, new algorithms for multi-level fluid descriptions have been developed. The dynamic refinement technique for Lattice Boltzmann schemes as well as the optimisation-based Lattice Boltzmann–Navier-Stokes scheme pave the way for (dynamic) continuum-to-statistical flow simulations. This induces new functionality available and applicable for existing sim-
Figure 18.2: Triple-scale simulation of channel flow.

ulation software. Besides, the methodology for hybrid LB–NS simulations also builds a starting point for more efficient CFD simulations, combining the features of both LB and NS solvers. Second, an approach to more standardised, yet flexible coupling software for hybrid molecular–continuum simulations has been developed. The arising software represents the first attempt to modularise and incorporate the functionalities for molecular–continuum schemes—transfer mechanisms for physical quantities, 2D and 3D support, parallelism. Similar to other framework-like approaches, the capabilities and the feasibility of the overall software concept are expected to be only completely proven in future by investigating new molecular–continuum applications and further coupling scheme implementations.

**Long-Term Perspective: Triple-Scale Simulation of Flows**

Based on the developed software, that is

- the spatially adaptive Lattice Boltzmann application in Peano,
- the coupling methodology for hybrid LB–NS simulations and
- the macro-micro-coupling tool for MD–LB (or similar molecular–continuum) schemes,

as well as the existing Navier-Stokes solver of Peano, a *triple-scale* simulation of flow systems is to become accessible in future, cf. Fig. 18.2: in this illustration of a channel flow, the bulk is resolved at Navier-Stokes level, the finest grid layers are solved by the Lattice Boltzmann method and the region very close to the channel wall is computed by molecular dynamics. For this perspective, the sequential and parallel efficiency of the involved simulation codes is expected to play a major role; this particularly holds for the molecular dynamics solver which allows to resolve the flow on time scales which are orders of magnitude smaller than in the Navier-Stokes simulation in this case.
Appendix

In the following, the matrices $B^{\text{eqim}}$, $B^{\text{ext}} \in \mathbb{R}^{Q \times D(D+1)/2}$ (cf. Sec. 12.2) are listed for the D3Q15, D3Q19 and D3Q27 model. Each of the $D(D+1)/2 = 6$ columns is related to one entry of the stress tensor. They are sorted as follows: $\tau_{xx}$, $\tau_{xy}$, $\tau_{xz}$, $\tau_{yy}$, $\tau_{yz}$, $\tau_{zz}$.

### D3Q15

\[
B^{\text{eqim}} = \begin{pmatrix}
B_0 & -\frac{1}{24} & -\frac{1}{24} & B_0 & -\frac{1}{24} & B_0 \\
B_0 & \frac{1}{24} & \frac{1}{24} & B_0 & -\frac{1}{24} & B_0 \\
B_1 & 0 & 0 & B_1 & 0 & B_2 \\
B_0 & \frac{1}{24} & -\frac{1}{24} & B_0 & \frac{1}{24} & B_0 \\
B_0 & -\frac{1}{24} & \frac{1}{24} & B_0 & \frac{1}{24} & B_0 \\
B_1 & 0 & 0 & B_2 & 0 & B_1 \\
B_2 & 0 & 0 & B_1 & 0 & B_1 \\
B_3 & 0 & 0 & B_3 & 0 & B_3 \\
B_2 & 0 & 0 & B_1 & 0 & B_1 \\
B_1 & 0 & 0 & B_2 & 0 & B_1 \\
B_0 & -\frac{1}{24} & \frac{1}{24} & B_0 & \frac{1}{24} & B_0 \\
B_0 & -\frac{1}{24} & \frac{1}{24} & B_0 & \frac{1}{24} & B_0 \\
B_1 & 0 & 0 & B_1 & 0 & B_2 \\
B_0 & \frac{1}{24} & \frac{1}{24} & B_0 & \frac{1}{24} & B_0 \\
B_0 & -\frac{1}{24} & -\frac{1}{24} & B_0 & \frac{1}{24} & B_0 \\
B_0 & -\frac{1}{24} & -\frac{1}{24} & B_0 & \frac{1}{24} & B_0 \\
\end{pmatrix}
\]

with

\[
B_0 = -\frac{g_1 + 4g_0}{6(g_{\sqrt{3}} + 12g_1 + 32g_0)}
\]

\[
B_1 = -4B_0
\]

\[
B_2 = -\frac{g_{\sqrt{3}} + 8g_1 + 16g_0}{6(g_{\sqrt{3}} + 12g_1 + 32g_0)}
\]

\[
B_3 = \frac{g_{\sqrt{3}} + 4g_1}{3(g_{\sqrt{3}} + 12g_1 + 32g_0)}
\]
### D3Q19

\[
B^{\text{opt.}} = \begin{pmatrix}
  B_0 & 0 & 0 & B_1 & -\frac{1}{12} & B_1 \\
  B_1 & 0 & -\frac{1}{12} & B_0 & 0 & B_1 \\
  B_2 & 0 & 0 & B_2 & 0 & B_3 \\
  B_1 & 0 & \frac{1}{12} & B_0 & 0 & B_1 \\
  B_0 & 0 & 0 & B_1 & \frac{1}{12} & B_1 \\
  B_1 & -\frac{1}{12} & 0 & B_1 & 0 & B_0 \\
  B_2 & 0 & 0 & B_3 & 0 & B_2 \\
  B_1 & \frac{1}{12} & 0 & B_1 & 0 & B_0 \\
  B_3 & 0 & 0 & B_2 & 0 & B_2 \\
  B_4 & 0 & 0 & B_4 & 0 & B_4 \\
  B_3 & 0 & 0 & B_2 & 0 & B_2 \\
  B_1 & \frac{1}{12} & 0 & B_1 & 0 & B_0 \\
  B_2 & 0 & 0 & B_3 & 0 & B_2 \\
  B_1 & -\frac{1}{12} & 0 & B_1 & 0 & B_0 \\
  B_0 & 0 & 0 & B_1 & \frac{1}{12} & B_1 \\
  B_1 & 0 & \frac{1}{12} & B_0 & 0 & B_1 \\
  B_2 & 0 & 0 & B_2 & 0 & B_3 \\
  B_1 & 0 & -\frac{1}{12} & B_0 & 0 & B_1 \\
  B_0 & 0 & 0 & B_1 & -\frac{1}{12} & B_1
\end{pmatrix}
\]

\[
B^{\text{opt.}} = \begin{pmatrix}
  \frac{1}{36} & 0 & 0 & -\frac{1}{13} & -\frac{1}{12} & -\frac{1}{13} \\
  -\frac{1}{13} & 0 & -\frac{1}{12} & \frac{1}{36} & 0 & -\frac{1}{13} \\
  \frac{1}{13} & 0 & 0 & \frac{1}{36} & 0 & -\frac{1}{9} \\
  -\frac{1}{13} & 0 & \frac{1}{13} & 0 & -\frac{1}{36} & 0 \\
  \frac{1}{36} & 0 & 0 & \frac{1}{13} & \frac{1}{12} & \frac{1}{13} \\
  -\frac{1}{13} & 0 & \frac{1}{13} & 0 & -\frac{1}{36} & 0 \\
  \frac{1}{36} & 0 & 0 & \frac{1}{13} & \frac{1}{12} & \frac{1}{13} \\
  -\frac{1}{13} & 0 & \frac{1}{13} & 0 & -\frac{1}{36} & 0 \\
  \frac{1}{36} & 0 & 0 & \frac{1}{13} & \frac{1}{12} & \frac{1}{13} \\
  -\frac{1}{13} & 0 & \frac{1}{13} & 0 & -\frac{1}{36} & 0 \\
  \frac{1}{36} & 0 & 0 & \frac{1}{13} & \frac{1}{12} & \frac{1}{13} \\
\end{pmatrix}
\]

with

\[
B_0 = \frac{-g_{\sqrt{2}g_0} + 2g_1^2 + 2g_1g_0}{3(g_{\sqrt{2}} + 8g_1 + 12g_0)(g_{\sqrt{2}} + 2g_1)}
\]

\[
B_1 = \frac{-2g_{\sqrt{2}g_0} + 8g_1g_0 + g_{\sqrt{2}g_1} + 4g_1^2}{6(g_{\sqrt{2}} + 8g_1 + 12g_0)(g_{\sqrt{2}} + 2g_1)}
\]

\[
B_2 = \frac{4g_{\sqrt{2}g_0} + 4g_1g_0 + g_{\sqrt{2}g_1}}{3(g_{\sqrt{2}} + 8g_1 + 12g_0)(g_{\sqrt{2}} + 2g_1)}
\]

\[
B_3 = \frac{-4g_{\sqrt{2}g_0} + 8g_1g_0 - g_1^2 - 6g_{\sqrt{2}g_1}}{6(g_{\sqrt{2}} + 8g_1 + 12g_0)(g_{\sqrt{2}} + 2g_1)}
\]

\[
B_4 = \frac{g_{\sqrt{2}} + 4g_1}{3(g_{\sqrt{2}} + 8g_1 + 12g_0)}
\]

(A.3)
$$B^{\text{prim.}} = \begin{pmatrix}
B_0 & -B_1 & -B_1 & B_0 & -B_1 & B_0 \\
B_2 & 0 & 0 & B_3 & -B_6 & B_3 \\
B_0 & B_1 & B_1 & B_0 & -B_1 & B_0 \\
B_3 & 0 & -B_6 & B_2 & 0 & B_3 \\
B_4 & 0 & 0 & B_4 & 0 & B_2 \\
B_3 & 0 & B_6 & B_2 & 0 & B_3 \\
B_0 & B_1 & -B_1 & B_0 & B_1 & B_0 \\
B_2 & 0 & 0 & B_3 & B_6 & B_3 \\
B_0 & -B_1 & B_1 & B_0 & B_1 & B_0 \\
B_3 & -B_6 & 0 & B_3 & 0 & B_2 \\
B_4 & 0 & 0 & B_5 & 0 & B_4 \\
B_3 & B_6 & 0 & B_3 & 0 & B_2 \\
B_4 & 0 & 0 & B_5 & 0 & B_4 \\
B_3 & -B_6 & 0 & B_3 & 0 & B_2 \\
B_0 & -B_1 & B_1 & B_0 & B_1 & B_0 \\
B_2 & 0 & 0 & B_3 & B_6 & B_3 \\
B_0 & B_1 & -B_1 & B_0 & B_1 & B_0 \\
B_3 & 0 & B_6 & B_2 & 0 & B_3 \\
B_4 & 0 & 0 & B_4 & 0 & B_2 \\
B_3 & 0 & -B_6 & B_2 & 0 & B_3 \\
B_0 & B_1 & B_1 & B_0 & -B_1 & B_0 \\
B_2 & 0 & 0 & B_3 & -B_6 & B_3 \\
B_0 & -B_1 & -B_1 & B_0 & -B_1 & B_0
\end{pmatrix}$$

$$B^{\text{alt.}} = \begin{pmatrix}
-\frac{1}{158} & -\frac{1}{12} & -\frac{1}{12} & -\frac{1}{158} & -\frac{1}{12} & -\frac{1}{158} \\
\frac{7}{36} & 0 & 0 & -\frac{1}{27} & -\frac{1}{15} & -\frac{1}{27} \\
-\frac{1}{158} & -\frac{1}{12} & -\frac{1}{12} & -\frac{1}{158} & -\frac{1}{12} & -\frac{1}{158} \\
-\frac{1}{27} & 0 & -\frac{1}{15} & -\frac{1}{3} & 0 & -\frac{1}{27} \\
\frac{7}{27} & 0 & 0 & -\frac{2}{27} & 0 & -\frac{4}{27} \\
-\frac{1}{27} & 0 & -\frac{1}{15} & -\frac{1}{3} & 0 & -\frac{1}{27} \\
-\frac{1}{158} & -\frac{1}{12} & -\frac{1}{12} & -\frac{1}{158} & -\frac{1}{12} & -\frac{1}{158} \\
\frac{7}{36} & 0 & 0 & -\frac{1}{27} & -\frac{1}{15} & -\frac{1}{27} \\
-\frac{1}{27} & -\frac{1}{15} & 0 & -\frac{1}{3} & 0 & -\frac{1}{27} \\
2 \frac{2}{27} & 0 & 0 & -\frac{4}{27} & 0 & -\frac{2}{27} \\
-\frac{1}{27} & -\frac{1}{15} & 0 & -\frac{1}{3} & 0 & -\frac{1}{27} \\
-\frac{4}{27} & 0 & 0 & \frac{2}{27} & 0 & \frac{2}{27} \\
-\frac{1}{158} & -\frac{1}{12} & -\frac{1}{12} & -\frac{1}{158} & -\frac{1}{12} & -\frac{1}{158} \\
\frac{7}{36} & 0 & 0 & -\frac{1}{27} & -\frac{1}{15} & -\frac{1}{27} \\
-\frac{1}{27} & 0 & -\frac{1}{15} & -\frac{1}{3} & 0 & -\frac{1}{27} \\
\frac{7}{27} & 0 & 0 & -\frac{2}{27} & 0 & -\frac{4}{27} \\
-\frac{1}{27} & 0 & -\frac{1}{15} & -\frac{1}{3} & 0 & -\frac{1}{27} \\
-\frac{1}{158} & -\frac{1}{12} & -\frac{1}{12} & -\frac{1}{158} & -\frac{1}{12} & -\frac{1}{158} \\
\frac{7}{36} & 0 & 0 & -\frac{1}{27} & -\frac{1}{15} & -\frac{1}{27} \\
-\frac{1}{27} & 0 & -\frac{1}{15} & -\frac{1}{3} & 0 & -\frac{1}{27} \\
\frac{7}{27} & 0 & 0 & -\frac{2}{27} & 0 & -\frac{4}{27} \\
-\frac{1}{27} & 0 & -\frac{1}{15} & -\frac{1}{3} & 0 & -\frac{1}{27} \\
-\frac{1}{158} & -\frac{1}{12} & -\frac{1}{12} & -\frac{1}{158} & -\frac{1}{12} & -\frac{1}{158}
\end{pmatrix}$$

(A.5)
with

\[ B_0 = -\frac{4g_1\sqrt{g_0} + g_1\sqrt{g_1} + 4g_1g_0}{6(g_0\sqrt{g_0} + 8g_1\sqrt{g_1} + 12g_0\sqrt{g_0} + 12g_1\sqrt{g_1} + 32g_0g_0 + 16g_1g_0)} \]

\[ B_1 = \frac{g_1}{12(g_0\sqrt{g_0} + 2g_1\sqrt{g_1})} \]

\[ B_2 = -\frac{(g_0\sqrt{g_0} + 2g_1\sqrt{g_1} - 2g_0\sqrt{g_0} - 4g_1\sqrt{g_1} - 12g_0g_1g_0 - 8g_1g_0)}{(3g_0\sqrt{g_0} + 16g_1\sqrt{g_1} + 24g_0\sqrt{g_0} + 12g_1\sqrt{g_1} + 32g_0g_0 + 32g_1g_0 + 10g_1\sqrt{g_1} + 12g_0\sqrt{g_0} + 24g_1\sqrt{g_1} + 80g_1g_0)} \]

\[ B_3 = \frac{(2g_0\sqrt{g_0} + 4g_1\sqrt{g_1} + 8g_0\sqrt{g_0} + 4g_1\sqrt{g_1} + 8g_0\sqrt{g_0} + 4g_1\sqrt{g_1})}{(6g_0\sqrt{g_0} + 16g_1\sqrt{g_1} + 24g_0\sqrt{g_0} + 12g_1\sqrt{g_1} + 32g_0g_0 + 32g_1g_0 + 10g_1\sqrt{g_1} + 12g_0\sqrt{g_0} + 24g_1\sqrt{g_1} + 80g_1g_0)} \]

\[ B_4 = \frac{(4g_0\sqrt{g_0} + 4g_1\sqrt{g_1} + 8g_0\sqrt{g_0} + 4g_1\sqrt{g_1} + 8g_0\sqrt{g_0} + 4g_1\sqrt{g_1})}{(3g_0\sqrt{g_0} + 16g_1\sqrt{g_1} + 24g_0\sqrt{g_0} + 12g_1\sqrt{g_1} + 32g_0g_0 + 32g_1g_0 + 10g_1\sqrt{g_1} + 12g_0\sqrt{g_0} + 24g_1\sqrt{g_1} + 80g_1g_0)} \]

\[ B_5 = \frac{(g_0\sqrt{g_0} + 4g_1\sqrt{g_1} + 6g_0\sqrt{g_0} + 4g_1\sqrt{g_1} + 8g_0\sqrt{g_0} + 4g_1\sqrt{g_1} + 6g_0\sqrt{g_0} + 4g_1\sqrt{g_1})}{(6g_0\sqrt{g_0} + 16g_1\sqrt{g_1} + 24g_0\sqrt{g_0} + 12g_1\sqrt{g_1} + 32g_0g_0 + 32g_1g_0 + 10g_1\sqrt{g_1} + 12g_0\sqrt{g_0} + 24g_1\sqrt{g_1} + 80g_1g_0)} \]

\[ B_6 = \frac{g_1}{12(g_0\sqrt{g_0} + 2g_1\sqrt{g_1})} \]

\[ B_7 = \frac{g_1\sqrt{g_0} + 4g_1\sqrt{g_1} + 4g_1g_0}{3(g_0\sqrt{g_0} + 8g_1\sqrt{g_1} + 12g_0\sqrt{g_0} + 12g_1\sqrt{g_1} + 32g_0g_0 + 16g_1g_0)} \]
Bibliography


176
BIBLIOGRAPHY


